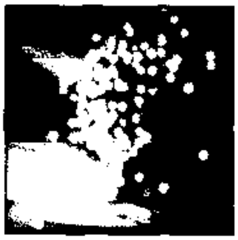
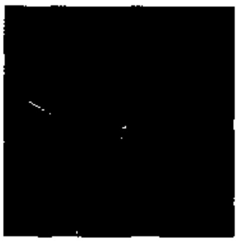
Tercera Edición

Fundamentos de programación

Algoritmos, Estructuras de datos y Objetos

Mc Graw Hill

Luis JOYANES AGUILAR

FUNDAMENTOS DE PROGRAMACIÓN

Algoritmos, estructuras de datos y objetos

Tercera edición

CONTENIDO

Prólogo a la tercera edición .... **xiii**

Prólogo a la segunda edición .... **xxvii**

PARTE I
Algoritmos y herramientas de programación

Capítulo 1. Computadoras y lenguajes de programación **3**

1.1. Organización de una computadora 4
1.2. *Hardware* 5
1.3. Dispositivos de almacenamiento de información 13
1.4. La computadora personal ideal para programación 18
1.5. El *Software* (los programas) 19
1.6. Los lenguajes de programación 21
1.7. Traductores de lenguaje 24
1.8. Historia de los lenguajes de programación 28
ACTIVIDADES DE PROGRAMACIÓN RESUELTAS 34
REVISIÓN DEL CAPÍTULO 36
Conceptos clave 36
Resumen 36
EJERCICIOS 37

Capítulo 2. Resolución de problemas con computadora y herramientas de programación **39**

2.1. Fases en la resolución de problemas 40
2.2. Programación modular 49
2.3. Programación estructurada 50
2.4. Concepto y características de algoritmos 52
2.5. Escritura de algoritmos 56
ACTIVIDADES DE PROGRAMACIÓN RESUELTAS 71
REVISIÓN DEL CAPÍTULO 79
Conceptos clave 79
Resumen 79
EJERCICIOS 79

Capítulo 3. Estructura general de un programa **83**

3.1. Concepto de programa 84
3.2. Partes constitutivas de un programa 84
3.3. Instrucciones y tipos de instrucciones 85
3.4. Elementos básicos de un programa 88
3.5. Datos, tipos de datos y operaciones primitivas 90
3.6. Constantes y variables 92
3.7. Expresiones 94
3.8. Funciones internas 102
3.9. La operación de asignación 103
3.10. Entrada y salida de información 106
3.11. Escritura de algoritmos/programas 107
ACTIVIDADES DE PROGRAMACIÓN RESUELTAS 112
REVISIÓN DEL CAPÍTULO 124
Conceptos clave 124
Resumen 125
EJERCICIOS 125

PARTE II
Programación estructurada: Algoritmos y estructuras de datos

Capítulo 4. Flujo de control I: Estructuras selectivas **131**

4.1. El flujo de côntrol de un programa 132
4.2. Estructura secuencial 132
4.3. Estructuras selectivas 135
4.4. Alternativa simple (`si-entonces/if-then`) 135
4.5. Alternativa múltiple (`según_sea, caso de/case`) 142
4.6. Estructuras de decisión anidadas (en escalera) 149
4.7. La sentencia `ir_a ("goto")` 153
ACTIVIDADES DE PROGRAMACIÓN RESUELTAS 156
REVISIÓN DEL CAPÍTULO 159
Conceptos clave 159
Resumen 160
EJERCICIOS 161

Capítulo 5. Flujo de control II: Estructuras repetitivas **163**

5.1. Estructuras repetitivas 164
5.2. Estructura `mientras ("while")` 166
5.3. Estructura `hacer-mientras ("do-while")` 172
5.4. Estructura `repetir ("repeat")` 174
5.5. Estructura `desde/para ("for")` 177
5.6. Salidas internas de los bucles 182
5.7. Sentencias de salto `interrumpir (break)` y `continuar (continue)` 183
5.8. Estructuras repetitivas anidadas 185
ACTIVIDADES DE PROGRAMACIÓN RESUELTAS 187
REVISIÓN DEL CAPÍTULO 201
Conceptos clave 201
Resumen 201
EJERCICIOS 202
REFERENCIAS BIBLIOGRÁFICAS 203

Capítulo 6. Subprogramas (subalgoritmos): Procedimientos y funciones **205**

6.1. Introducción a los subalgoritmos o subprogramas 206
6.2. Funciones 207

6.3. Procedimientos (subrutinas) 215
6.4. Ámbito: variables locales y globales 220
6.5. Comunicación con subprogramas: paso de parámetros 223
6.6. Funciones y procedimientos como parámetros 232
6.7. Los efectos laterales 234
6.8. Recursión (recursividad) 235
ACTIVIDADES DE PROGRAMACIÓN RESUELTAS 239
REVISIÓN DEL CAPÍTULO 244
Conceptos clave 244
Resumen 244
EJERCICIOS 245

Capítulo 7. Estructuras de datos I: (arrays y estructuras) 247

7.1. Introducción a las estructuras de datos 248
7.2. Arrays unidimensionales: los vectores 249
7.3. Operaciones con vectores 252
7.4. Arrays de varias dimensiones 258
7.5. Arrays multidimensionales 262
7.6. Almacenamiento de arrays en memoria 263
7.7. Estructuras *versus* registros 266
7.8. Arrays de estructuras 268
ACTIVIDADES DE PROGRAMACIÓN RESUELTAS 270
REVISIÓN DEL CAPÍTULO 282
Conceptos clave 282
Resumen 282
EJERCICIOS 283

Capítulo 8. Las cadenas de caracteres 285

8.1. Introducción 286
8.2. El juego de caracteres 286
8.3. Cadena de caracteres 290
8.4. Datos tipo carácter 291
8.5. Operaciones con cadenas 294
8.6. Otras funciones de cadenas 299
ACTIVIDADES DE PROGRAMACIÓN RESUELTAS 302
REVISIÓN DEL CAPÍTULO 307
Conceptos clave 307
Resumen 308
EJERCICIOS 308

Capítulo 9. Archivos (ficheros) 311

9.1. Noción de archivo (fichero): estructura jerárquica 312
9.2. Conceptos y definiciones = terminología 314
9.3. Soportes secuenciales y direccionables 317
9.4. Organización de archivos 317
9.5. Operaciones sobre archivos 321
9.6. Gestión de archivos 325
9.7. Borrar archivos 329
9.8. Flujos 329
9.9. Mantenimiento de archivos 330
9.10. Procesamiento de archivos secuenciales (algoritmos) 331
9.11. Archivos de texto 338

9.12. Procesamiento de archivos directos (algoritmos) 339
9.13. Procesamiento de archivos secuenciales indexados 348
ACTIVIDADES DE PROGRAMACIÓN RESUELTAS 349
REVISIÓN DEL CAPÍTULO 356
Conceptos clave 356
Resumen 356
EJERCICIOS 357

Capítulo 10. Ordenación, búsqueda e intercalación 359

10.1. Introducción 360
10.2. Ordenación 361
10.3. Búsqueda 379
10.4. Intercalación 396
ACTIVIDADES DE PROGRAMACIÓN RESUELTAS 398
REVISIÓN DEL CAPÍTULO 411
Conceptos clave 411
Resumen 411
EJERCICIOS 412

Capítulo 11. Ordenación, búsqueda y fusión externa (archivos) 413

11.1. Introducción 414
11.2. Archivos ordenados 414
11.3. Fusión de archivos 415
11.4. Partición de archivos 418
11.5. Clasificación de archivos 423
ACTIVIDADES DE PROGRAMACIÓN RESUELTAS 432
REVISIÓN DEL CAPÍTULO 436
Conceptos clave 436
Resumen 436
EJERCICIOS 437

Capítulo 12. Estructuras dinámicas lineales de datos (pilas, colas y listas enlazadas) 439

12.1. Introducción a las estructuras de datos 440
12.2. Listas 441
12.3. Listas enlazadas 443
12.4. Procesamiento de listas enlazadas 447
12.5. Listas circulares 462
12.6. Listas doblemente enlazadas 463
12.7. Pilas 465
12.8. Colas 474
12.9. Doble cola 482
ACTIVIDADES DE PROGRAMACIÓN RESUELTAS 483
REVISIÓN DEL CAPÍTULO 492
Conceptos clave 492
Resumen 492
EJERCICIOS 493

Capítulo 13. Estructura de datos no lineales (árboles y grafos) 495

13.1. Introducción 496
13.2. Árboles 496
13.3. Árbol binario 498
13.4. Árbol binario de búsqueda 511

13.5. Grafos 523
ACTIVIDADES DE PROGRAMACIÓN RESUELTAS 529
REVISIÓN DEL CAPÍTULO 534
Conceptos clave 534
Resumen 535
EJERCICIOS 535

Capítulo 14. Recursividad 537

14.1. La naturaleza de la recursividad 538
14.2. Recursividad directa e indirecta 542
14.3. Recursión *versus* iteración 546
14.4. Recursión infinita 549
14.5. Resolución de problemas complejos con recursividad 553
REVISIÓN DEL CAPÍTULO 568
Conceptos clave 568
Resumen 568
EJERCICIOS 569

Parte III
Programación Orientada a Objetos (POO)

Capítulo 15. Conceptos fundamentales de orientación a objetos 575

15.1. ¿Qué es programación orientada a objetos? 576
15.2. Un mundo de objetos 580
15.3. Comunicaciones entre objetos: los mensajes 586
15.4. Estructura interna de un objeto 589
15.5. Clases 591
15.6. Herencia 593
15.7. Sobrecarga 600
15.8. Ligadura dinámica 602
15.9. Objetos compuestos 603
15.10. Reutilizacion con orientación a objetos 607
15.11. Polimorfismo 607
15.12. Terminología de orientación a objetos 608
REVISIÓN DEL CAPÍTULO 609
Conceptos clave 609
Resumen 609
EJERCICIOS 610

Capítulo 16. Diseño de clases y objetos: Representaciones gráficas en UML 613

16.1. Diseño y representación gráfica de objetos en UML 614
16.2. Diseño y representación gráfica de clases en UML 623
16.3. Declaración de objetos de clases 632
16.4. Constructores 641
16.5. Destructores 646
16.6. Implementación de clases en C++ 647
16.7. Recolección de basura 650
REVISIÓN DEL CAPÍTULO 651
Conceptos clave 651
Resumen 652
EJERCICIOS 653
LECTURAS RECOMENDADAS 655

Capítulo 17. Relaciones: Asociación, generalización, herencia **657**

17.1. Relaciones entre clases 658
17.2. Asociaciones 658
17.3. Agregaciones 660
17.4. Jerarquía de clases: generalización y especialización (relación es-un) 662
17.5. Herencia: clases derivadas 667
17.6. Tipos de herencia 675
17.7. Herencia múltiple 680
17.8. Ligadura 684
17.9. Polimorfismo 686
17.10. Ligadura dinámica frente a ligadura estática 690
REVISIÓN DEL CAPÍTULO 691
Conceptos clave 691
Resumen 691
EJERCICIOS 692

Parte IV
Metodología de la programación y desarrollo de software

Capítulo 18. Resolución de problemas y desarrollo de software: Metodología de la programación **697**

18.1. Abstracción y resolución de problemas 698
18.2. El ciclo de vida del software 701
18.3. Fase de análisis: requisitos y especificaciones 703
18.4. Diseño 704
18.5. Implementación (codificación) 706
18.6. Pruebas e integración 706
18.7. Mantenimiento 707
18.8. Principios de diseño de sistemas de software 708
18.9. Estilo de programación 713
18.10. La documentación 718
18.11. Depuración 721
18.12. Diseño de algoritmos 724
18.13. Pruebas (testing) 724
18.14. Eficiencia 728
18.15. Transportabilidad 730
REVISIÓN DEL CAPÍTULO 731
Conceptos clave 731
Resumen 731

APÉNDICES

A. Especificaciones de lenguaje algorítmico UPSAM 2.0 .... **735**

B. Prioridad de operadores **755**

C. Códigos ASCII y Unicode **757**

D. Guía de sintaxis del lenguaje C **763**

E. Guía de sintaxis del lenguaje C++ **791**

F. Guía de sintaxis del lenguaje Java 2 .... **843**

G. Guía de sintaxis del lenguaje C# **893**

H. Palabras reservadas: C++, Java, C# **927**

I. Codificación de algoritmos en lenguajes de programación: Pascal, Fortran y Modula-2 .... **931**

J. Guía de sintaxis de Pascal (Borland Turbo Pascal 7.0 y Delphi) **951**

K. Recursos de programación: Libros, revistas, web, lecturas recomendadas **975**

Índice alfabético **989**

PRÓLOGO A LA TERCERA EDICIÓN

La informática y las ciencias de la computación en los primeros años del siglo XXI vienen marcadas por los avances tecnológicos de la pasada década. Los más de veinte años de vida del computador personal (PC) y los más de cincuenta años de la informática/computación tradicional vienen acompañados de cambios rápidos y evolutivos en las disciplinas clásicas. El rápido crecimiento del mundo de las redes y, en consecuencia, la World Wide Web hacen revolucionarios a estos cambios y afectan al cuerpo de conocimiento de los procesos educativos y profesionales.

Así, como declara ACM en su informe final (15 de diciembre de 2001) CC2001 Computer Science, la formación en carreras de informática, ciencias de la computación o ingeniería de sistemas deberá prestar especial importancia a temas tales como:

- Algoritmos y estructuras de datos.
- La World Wide Web y sus aplicaciones.
- Las tecnologías de red y en especial aquellas basadas en TCP/IP.
- Gráficas y multimedia.
- Sistemas empotrados.
- Bases de datos relacionales.
- Inteoperabilidad.
- Programación orientada a objetos.
- Interacción Hombre-Máquina.
- ...

ACM, velando porque sus miembros —al fin y al cabo, representantes de la comunidad informática mundial— sigan los progresos científicos y, en consecuencia, culturales y sociales derivados de las innovaciones tecnológicas, ha trabajado durante muchos años en un nuevo modelo curricular de la carrera de ingeniero informático o ingeniero de sistemas (computer sciences) y a finales del 2001 publicó su anteproyecto de currículo profesional (informe CC2001). El cuerpo de conocimiento incluído en este informe contempla una estructura con 14 grupos de conocimiento que van desde las Estructuras Discretas a la Ingeniería de Software pasando por Fundamentos de Programación (*Programming Fundamentals*, **PF**). En nuestro caso y para reconocimiento de las citadas palabras, nos cabe el honor de que nuestra obra, que ha cumplido recientemente QUINCE AÑOS DE VIDA, tenga

el mismo título en español (**Fundamentos de Programación, Programming Fundamentals**) que uno de los 14 grupos de conocimiento ahora recomendados como disciplina fundamental por la ACM dentro de su currículo de *Computer Science*. Así, en el citado currículo se incluyen descriptores tales como: *construcciones de programación fundamentales, algoritmos y resolución de problemas, estructuras de datos fundamentales* y *recursividad o recursión.*

También los planes de estudios de ingeniería informática en España (superior y técnicas de sistemas o de gestión) y de otras ingenierías (telecomunicaciones, industriales, etc.) y de ingeniería de sistemas en Latinoamérica, incluyen asignaturas tales como *Metodología de la Programación, Fundamentos de Programación* o *Introducción a la Programación.*

Del estudio comparado de las citadas recomendaciones curriculares, así como de planes de estudios conocidos de carreras de ingeniería de sistemas y licenciaturas en informática de universidades latinoamericanas, hemos llegado a la consideración de que la iniciación de un estudiante de ingeniería informática o de ingeniería de sistemas en las técnicas de programación del siglo XXI requiere no sólo del aprendizaje clásico del diseño de algoritmos y de la comprensión de las técnicas orientadas a objetos, sino un método de transición hacia tecnologías de Internet. Por otra parte, un libro dirigido a los primeros cursos de introducción a la programación exige no sólo la elección de un lenguaje de programación adecuado si no, y sobre todo, «proporcionar al lector las herramientas para desarrollar programas correctos, eficientes, bien estructurados y con estilo, que sirvan de base para la construcción de unos fundamentos teóricos y prácticos que le permitan continuar con éxito sus estudios de los cursos superiores de su carrera, así como su futura especialización en ciencias e ingeniería». En consecuencia y de modo global, la obra pretende enseñar técnicas de **análisis, diseño y construcción de algoritmos, estructuras de datos** y **objetos**, así como reglas para la escritura de **programas, eficientes tanto estructurados, fundamentalmente, como orientados a objetos**. De modo complementario, y no por ello menos importante, se busca también enseñar al alumno técnicas de abstracción que le permitan resolver los problemas de programación del modo más sencillo y racional pensando no sólo en el aprendizaje de reglas de sintaxis y construcción de programas, sino, y sobre todo, aprender a pensar para conseguir la resolución del problema en cuestión de forma clara, eficaz y fácil de implementar en un lenguaje de programación y su ejecución posterior en una computadora u ordenador.

OBJETIVOS DEL LIBRO

El libro pretende enseñar a programar utilizando conceptos fundamentales, tales como:

1. *Algoritmos* (conjunto de instrucciones programadas para resolver una tarea específica).
2. *Datos* (una colección de datos que se proporcionan a los algoritmos que se han de ejecutar para encontrar una solución: los datos se organizarán en *estructuras de datos*).
3. *Objetos* (conjunto de datos y algoritmos que los manipulan, encapsulados en un tipo de dato conocido como objeto).
4. *Clases* (tipos de objetos con igual estado y comportamiento, o dicho de otro modo, los mismos atributos y operaciones).
5. *Estructuras de datos* (conjunto de organizaciones de datos para tratar y manipular eficazmente datos homogéneos y heterogéneos).
6. *Temas avanzados* (recursividad, métodos avanzados de ordenación y búsqueda, relaciones entre clases, etc.)

Los dos primeros aspectos, algoritmos y datos, han permanecido invariables a lo largo de la corta historia de la informática/computación, pero la *interrelación* entre ellos sí que ha variado y continuará haciéndolo. Esta interrelación se conoce como *paradigma de programación.*

En el paradigma de programación *procedimental (procedural* o *por procedimientos*) un problema se modela directamente mediante un conjunto de algoritmos. Por ejemplo, la nómina de una empresa o la gestión de ventas de un almacén se representan como una serie de procedimientos que manipulan datos. Los datos se almacenan separadamente y se accede a ellos o bien mediante una posición global o mediante parámetros en los procedimientos. Tres lenguajes de programación clásicos, FORTRAN, Pascal y C, han representado el arquetipo de la programación *procedimental*, también relacionada estrechamente y —normalmente— conocida como **programación estructurada**. La programación con soporte en C y Pascal proporciona el paradigma *procedimental* tradicional con un énfasis en funciones, plantillas de funciones y algoritmos genéricos.

En la década de los ochenta, el enfoque del diseño de programas se desplazó desde el paradigma *procedimental* al *orientado a objetos* apoyado en los tipos abstractos de datos (TAD). En este paradigma un problema se modela un conjunto de abstracciones de datos (tipos de datos) conocidos como *clases*. Las clases contienen un conjunto de instancias o ejemplares de la misma que se denominan *objetos*, de modo que un programa actúa como un conjunto de objetos que se relacionan entre sí. La gran diferencia entre ambos paradigmas reside en el hecho de que los algoritmos asociados con cada clase se conocen como *interfaz pública* de la clase y los datos se almacenan privadamente dentro de cada objeto, de modo que el acceso a los datos está oculto al programa general y se gestionan a través de la interfaz.

Así pues, en resumen, los objetivos fundamentales de esta obra son: aprendizaje y formación en *algoritmos y programación estructurada, estructuras de datos y programación orientada a objetos*. Evidentemente, la mejor forma de aprender a programar es con la ayuda de un lenguaje de programación. Apoyados en la experiencia de nuestras dos primeras ediciones y en los resultados conseguidos con nuestros alumnos y lectores, hemos seguido apostando por utilizar un lenguaje algorítmico —pseudolenguaje— que apoyado en un *pseudocódigo (seudocódigo)* en español nos permitiera enseñar al alumno las técnicas y reglas de programación y que su aprendizaje fuese rápido y gradual. Naturalmente, además del pseudocódigo hemos utilizado las otras herramientas de programación clásicas y probadas como los diagramas de flujo o los diagramas N-S.

En esta tercera edición hemos seguido utilizando el mismo lenguaje algorítmico al que le hemos añadido las construcciones y estructuras necesarias para incorporar en sus especificaciones las técnicas orientadas a objetos. Así, hemos seguido utilizando nuestro lenguaje inspirado en los lenguajes estructurados por excelencia, C, Pascal y FORTRAN, y le hemos añadido las propiedades de los lenguajes orientados a objetos tales como C++, Java y C#, en la mejor armonía posible y con unas especificaciones que hemos incluido en el Apéndice A, como ya hiciéramos también en las dos primeras ediciones.

EL LIBRO COMO HERRAMIENTA DOCENTE

En el contenido de la obra hemos tenido en cuenta no sólo las directrices de los planes de estudio españoles de ingeniería informática (antigua licenciatura en informática), ingeniería técnica en informática y licenciatura en ciencias de la computación, sino también de ingenierías, tales como industriales, telecomunicaciones, agrónomos o minas, o las más recientes incorporadas, en España, como ingeniería en geodesia, ingeniería química o ingeniería telemática. Nuestro conocimiento del mundo educativo iberoamericanao nos ha llevado a pensar también en las carreras de ingeniería de sistemas computacionales y las licenciaturas en informática y en sistemas de información, como se las conoce en Latinoamérica.

El contenido del libro se ha escrito pensando en un posible desarrollo de dos cuatrimestres o semestres o en un año completo, y siguiendo los descriptores (temas centrales) recomendados por el Consejo de Universidades de España para los planes de estudio de **Ingeniería en Informática, Ingeniería Técnica en Informática** e **Ingeniería Técnica en Informática**, y los planes de estudios de

Ingeniería de sistemas y **Licenciaturas en Informática** con el objeto de poder ser utilizado también por los estudiantes de primeros semestres de estas carreras. Desde el punto de vista de asignaturas, se pretende que el libro pueda servir para asignaturas tales como *Introducción a la Programación, Fundamentos de Programación* y *Metodología de la Programación*, y también, si el lector o el maestro/profesor lo consideran oportuno, para cursos de *Introducción a Estructuras de Datos* y/o a *Programación Orientada a Objetos.*

No podíamos dejar de lado las recomendaciones de la más prestigiosa organización de informáticos del mundo, ACM, anteriormente citada. Se estudió en su momento los *Curricula* de *Computer Science* vigentes durante el largo periodo de elaboración de esta obra (68, 78 y 91), pero siguiendo la evolución del último *curricula*, por lo que tras su publicación el 15 de diciembre de 2001 del *«Computing Curricula 2001 Computer Science»* estudiamos el citado currículo y durante la escritura de nuestra tercera edición consideramos cómo introducir también sus directrices más destacadas; como lógicamente no se podía seguir todas las directrices de su cuerpo de conocimiento al pie de la letra, analizamos en profundidad las unidades más acordes con nuestros planes de estudios: *Programming Fundamentals (PF), Algorithms and Complexity (AL)* y *Programming Languages (PL).* Por suerte nuestra obra incorporaba la mayoría de los temas importantes recomendados en las tres citadas unidades de conocimiento, en gran medida de la unidad PF, y temas específicos de programación orientado a objetos y de análisis de algoritmos de las unidades PL y AL.

El contenido del libro abarca los citados programas y comienza con la introducción a los algoritmos y a la programación, para llegar a estructuras de datos y programación orientada a objetos. Por esta circunstancia la estructura del curso no ha de ser secuencial en su totalidad sino que el profesor/maestro y el alumno/lector podrán estudiar sus materias en el orden que consideren más oportuno. Esta es la razón principal por la cual el libro se ha organizado en cuatro partes y en seis apéndices.

Se trata de describir los *dos paradigmas* más populares en el mundo de la programación: el *procedimental* y el *orientado a objetos*. Los cursos de programación en sus niveles inicial y medio están evolucionando para aprovechar las ventajas de nuevas y futuras tendencias en ingeniería de software y en diseño de lenguajes de programación, específicamente diseño y programación orientada a objetos. Numerosas facultades y escuelas de ingenieros, junto con la **nueva formación profesional** (*ciclos formativos de nivel superior)* en España y en Latinoamérica, están introduciendo a sus alumnos en la programación orientada a objetos, inmediatamente después del conocimiento de la programación estructurada, e incluso —en ocasiones antes—. Por esta razón, una metodología que se podría seguir sería impartir un curso de *algoritmos e introducción a la programación* (parte I) seguido de *estructuras de datos* (parte II) y luego seguir con un segundo nivel de *programación avanzada y programacion orientada a objetos* (partes II, III y IV) que constituyen las cuatro partes del libro. De modo complementario, si el alumno o el profesor/maestro lo desea se puede practicar con algún lenguaje de programación estructurado u orientado a objetos, ya que pensando en esa posibilidad se ha ampliado el número de los apéndices del libro para incluir guías de sintaxis de los lenguajes de programación más populares hoy día, C, Pascal, C++, Java y C e incluso se han mantenido resúmenes de guías de sintaxis de FORTRAN y Modula-2.

Uno de los temas más debatidos en la educación en informática o en ciencias de la computación (*Computer Sciences*) es el rol de la programación en el currículo introductorio. A través de la historia de la disciplina —como fielmente reconoce en la introducción del Capítulo 7 relativo a cursos de introducción, ACM en su Computing Curricula 2001 [ACM91]— la mayoría de los cursos de introducción a la informática se han centrado principalmente en el desarrollo de habilidades o destrezas de programación. La adopción de un curso de introducción a la programación proviene de una serie de factores prácticos e históricos incluyendo los siguientes:

- La programación es una técnica esencial que debe ser dominada por cualquier estudiante de informática. Su inserción en los primeros cursos de la carrera asegura que los estudiantes tengan la facilidad necesaria con la programación para cuando se matriculan en los cursos de nivel intermedio y avanzado.

- La informática no se convirtió en una disciplina académica hasta después que la mayoría de las instituciones ha desarrollado un conjunto de cursos de programación introductorias que sirvan a una gran audiencia.
- El modelo de aprendizaje en programación siguió desde el principio las tempranas recomendaciones del Currículo 68 [ACM68] que comenzaba con un curso denominado «Introducción a la Computación», en la que la abrumadora mayoría de los temas estaban relacionados con la programación. Con posterioridad, y en el currículum del 78 de la ACM [ACM78] definía a estos cursos como «Introducción a la Programación» y se les denominó CS1 y CS2; hoy día se le sigue denominando así por la mayoría de los profesores y universidades que seguimos criterios emanados de la ACM.

La fluidez en un lenguaje de programación es prerrequisito para el estudio de las ciencias de la computación. Ya en 1991 el informe CC1991 de la ACM reconocía la exigencia del conocimiento de un lenguaje de programación.

- Los programas de informática deben enseñar a los estudiantes cómo usar al menos bien un lenguaje de programación. Además, recomendamos que los programas en informática deben enseñar a los estudiantes a ser competentes en lenguajes y que hagan uso de al menos dos paradigmas de programación. Como consecuencia de estas ideas, el currículo 2001 de la ACM contempla la necesidad de conceptos y habilidades que son fundamentales en la práctica de la programación con independencia del paradigma subyacente. Como resultado de este pensamiento, el área de Fundamentos de Programación incluye unidades sobre conceptos de programación, estructuras de datos básicas y procesos algorítmicos. Además de estas unidades fundamentales se requieren conceptos básicos pertenecientes a otras áreas, como son Lenguajes de Programación (PL, *Programming Languages*) y de Ingeniería de Software (SE, *Software Engineering*).

Los temas fundamentales que considera el currículo 2001 son:

- Construcciones fundamentales de programación.
- Algoritmos y resolución de problemas.
- Estructuras de datos fundamentales.
- Recursión o recursividad.
- Programación controlada por eventos.

De otras áreas, Lenguajes de Programación (PL), destacar:

- Revisión de lenguajes de programación.
- Declaraciones y tipos.
- Mecanismos de abstracción.
- Programación orientada a objetos.

Y de Ingeniería de Software (SE):

- Diseño de software.
- Herramientas y entornos de software.
- Requisitos y especificaciones de software.

Este libro se ha escrito pensando en que pudiera servir de referencia y guía de estudio para un primer curso de *introducción a la programación*, con una segunda parte que, a su vez, sirviera como continuación y para un posible segundo curso. de *introducción a las estructuras de datos* y a la *programación orientada a objetos* El objetivo final que busca es no sólo describir la sintaxis de un lenguaje

de programación, sino, y sobre todo, mostrar las características más sobresalientes del lenguaje algorítmico y a la vez enseñar técnicas de programación estructurada y orientada a objetos. Así pues, los objetivos fundamentales son:

- Énfasis fuerte en el análisis, construcción y diseño de programas.
- Un medio de resolución de problemas mediante técnicas de programación.
- Una introducción a la informática y a las ciencias de la computación usando una herramienta de programación denominada pseudocódigo.
- Aprendizaje de técnicas de construcción de programas estructurados y una iniciación a los programas orientados a objetos.

Así se tratará de enseñar las técnicas clásicas y avanzadas de programación estructurada, junto con técnicas orientadas a objetos. La programación orientada a objetos no es la panacea universal de un programador del siglo XXI, pero le ayudará a realizar tareas que, de otra manera, serían complejas y tediosas.

El contenido del libro trata de proporcionar soporte a un año académico completo (dos semestres o cuatrimestres), alrededor de 24 a 32 semanas, dependiendo lógicamente de su calendario y planificación. Los diez primeros capítulos pueden comprender el primer semestre y los restantes capítulos pueden impartirse en el segundo semestre. Lógicamente la secuencia y planificación real dependerá del maestro o profesor que marcará y señalará semana a semana la progresión que él considera lógica. Si usted es un estudiante autodidacta, su propia progresión vendrá marcada por las horas que dedique al estudio y al aprendizaje con la computadora, aunque no debe variar mucho del ritmo citado al principio de este párrafo.

EL LENGUAJE ALGORÍTMICO DE PROGRAMACIÓN UPSAM 2.0

Los cursos de introducción a la programación se apoyan siempre en un lenguaje de programación o en un pseudolenguaje sustentado sobre un pseudocódigo. En nuestro caso optamos por esta segunda opción: el pseudolenguaje con la herramienta del pseudocódigo en español o castellano. Desde la aparición de la primera edición, allá por finales de los años ochenta, apostamos fundamentalmente por el pseudocódigo para que junto con los diagramas de flujo y diagramas N-S nos ayudara a explicar técnicas de programación a nuestros alumnos. Con ocasión de la publicación de la segunda edición (año 1996) no sólo seguimos apostando otra vez, y fundamentalmente, por el pseudocódigo sino que juntamos los trabajos de todos los profesores del Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software de la Facultad de Informática y Escuela Universitaria de Informática de la Universidad Pontificia de Salamanca en el campus de Madrid y le dimos forma a la versión 1.0 del lenguaje UPSAM[1].

La versión 1.0 de UPSAM se apoyaba fundamentalmente en los lenguajes de programación Pascal (Turbo Pascal) y C, con referencias a FORTRAN, COBOL y BASIC, y algunas ideas del lenguaje C++ que comenzaba ya a ser un estandar en el mundo de la programación. Java nació en 1995 mientras el libro estaba en imprenta, aunque la edición de nuestra obra se publicó en 1996. Por ello el lenguaje algorítmico sólo contenía reglas de sintaxis y técnicas de programación estructurada. Sin embargo, en la segunda mitad de los noventa, C++ se acabó por imponer como estándar en el mundo de la programación, y Java iba asentándose como lenguaje de programación para la Web e Internet, por lo que la nueva especificación algorítmica debería contemplar estos lenguajes emergentes, consolidado en el caso de C++.

---

[1] En el prólogo de la segunda edición (incluido tambien en esta tercera edición) se referenciaban los nombres de todos los profesores que habíamos intervenido en la redacción de la especificación de la versión 1.0 y, por consiguiente, figuraban como autores de dicha especificación.

En estos dos años de la década, el 01 y el 02, y en el tercer año que ya estamos, Java no sólo se ha constituido como el lenguaje universal para Internet sino que también se está ya utilizando para cursos de introducción a la programación y sobre todo para cursos de programación orientada a objetos. También al principio del año 2000, Microsoft sacó al mercado C#, y cada vez más comienza a difundirse y parece que a gran velocidad. Por todo ello, la nueva edición del lenguaje algorítmico, la 2.0, se ha inspirado fundamentalmente en C y C++, así como en Java y C#, aunque hemos intentado no olvidar sus viejos orígenes apoyados en Pascal y algo menos en FORTRAN y COBOL.

Así pues, la versión 2.0 del lenguaje UPSAM que en el año 2003 presentamos se apoya en sus quince años de vida (originalmente se conocía como versión UPS) ha sido construída como una mejora de la versión 1.0 a la cual hemos añadido sobre todo una guía de especificaciones y sintaxis para programación orientada a objetos, cuyo soporte han sido —fundamentalmente— los lenguajes C++ y Java.

En esta nueva versión hemos trabajado de nuevo todos los profesores del área de programación de la Universidad Pontificia de Salamanca en el campus de Madrid pero, de un modo muy especial, los profesores compiladores de todas las propuestas de nuestros compañeros hemos sido además del autor de esta obra, y especialmente, los profesores **Luis Rodríguez Baena**, **Víctor Martín García** y **Matilde Fernández Azuela**. Destacar también por haber contribuido a la revisión de esta edición y como siempre la colaboración de los profesores **Joaquín Sánchez Abeger**, **Lucas Sánchez García**, **Ángel Hermoso** e **Ignacio Zahonero Martínez** y restantes compañeros del departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software de la Facultad de Informática y Escuela Universitaria de Informática del campus de Madrid, que han realizado excelentes aportaciones para esta nueva edición.

Deseo destacar de modo muy especial la gran aportación práctica y de investigación realizada a UPSAM 1.0 y a nuestra nueva versión UPSAM 2.0 de los profesores **María Luisa Díez Plata** y **Enrique Torres Franco** —profesores de Procesadores de Lenguajes (Compiladores) de nuestra Facultad de Informática— que durante numerosos cursos académicos —y siguen en la actualidad— han implementado partes del lenguaje en un traductor de lenguaje en las prácticas y proyectos de compiladores de los alumnos de la citada asignatura de cuarto curso (séptimo y octavo semestre), en algunos casos ya funcionando como prototipos. Al profesor Oscar Sanjuan por sus diseños gráficos en UML.

CARACTERÍSTICAS IMPORTANTES DEL LIBRO

Fundamentos de programación, tercera edición, utiliza en cada capítulo los siguientes elementos clave para conseguir obtener el mayor rendimiento del material incluido en sus diferentes capítulos:

- **Objetivos.** Enumera los conceptos y técnicas que el lector y los estudiantes aprenderán en el capítulo. Su lectura ayudará a los estudiantes a determinar si se han cumplido estos objetivos después de terminar el capítulo.
- **Contenido.** Índice completo del capítulo que facilita la lectura y la correcta progresión en la lectura y comprensión de los diferentes temas que se exponen posteriormente.
- **Introducción.** Abre el capítulo con una breve revisión de los puntos y objetivos más importantes que se tratarán y todo aquello que se puede esperar del mismo.
- **Descripción del capítulo.** Explicación usual de los apartados correspondientes del capítulo. En cada capítulo se incluyen ejemplos y ejercicios resueltos. Los listados de los programas completos o parciales se escriben en letra «courier» con la finalidad principal de que puedan ser identificados fácilmente por el lector. Todos ellos han sido probados para facilitar la práctica del lector/alumno.
- **Conceptos clave.** Enumera los términos de computación, informáticos y de programación más notables que se han descrito o tratado en el capítulo.

- **Resumen del capítulo.** Revisa los temas importantes que los estudiantes y lectores deben comprender y recordar. Busca también ayudar a reforzar los conceptos clave que se han aprendido en el capítulo.
- **Ejercicios.** Al final de cada capítulo se proporciona a los lectores una lista de ejercicios sencillos de modo que le sirvan de oportunidad para que puedan medir el avance experimentado mientras leen y siguen —en su caso— las explicaciones del profesor relativas al capítulo.
- **Problemas.** En muchos capítulos se incluyen enunciados de problemas propuestos para realizar por el alumno y que presentan una mayor dificultad que los ejercicios antes planteados. Se suelen incluir una serie de actividades y proyectos de programación que se le proponen al lector como tarea complementaria de los ejercicios.

A lo largo de todo el libro se incluyen una serie de recuadros —sombreados o no— que ofrecen al lector consejos, advertencias y reglas de uso del lenguaje y de técnicas de programación, con la finalidad de que puedan ir asimilando conceptos prácticos de interés que les ayuden en el aprendizaje y construcción de programas eficientes y de fácil lectura.

- **Recuadro.** Conceptos importantes que el lector debe considerar durante el desarrollo del capítulo.
- **Consejo.** Ideas, sugerencias, recomendaciones, ... al lector, con el objetivo de obtener el mayor rendimiento posible del lenguaje y de la programación.
- **Precaución.** Advertencia al lector para que tenga cuidado al hacer uso de los conceptos incluidos en el recuadro adjunto.
- **Nota.** Normas o ideas que el lector debe seguir preferentemente en el diseño y construcción de sus programas.

ORGANIZACIÓN DEL LIBRO

El libro se ha dividido en cuatro partes a efectos de organización para su lectura y estudio gradual. Dado que el conocimiento es acumulativo, se comienza en los primeros capítulos con conceptos conceptuales y prácticos, y se avanza de modo progresivo hasta llegar a las técnicas avanzadas y a una introducción a la ingeniería de software que intentan prepara al lector/estudiante para sus estudios posteriores. Como complemento y ayuda al lector durante sus estudios y para su posterior formación profesional, se han incluído una gran cantidad de apéndices que incluyen fundamentalmente guías de sintaxis de los lenguajes de programación más populares con el objetivo de facilitar la implementación de los algoritmos en el lenguaje de programación elegido para «dialogar» con la computadora. Así mismo, y para ayudar a la preparación del aprendizaje, lecturas y estudios futuros, se han incluido una amplia guía de recursos de programación (libros, revistas y sitios Web «URLs») que hemos consultado en la elaboración de nuestra obra y seguimos consultando también en nuestra vida profesional.

PARTE I. ALGORITMOS Y HERRAMIENTAS DE PROGRAMACIÓN

Esta parte es un primer curso de programación para alumnos principiantes en asignaturas de introducción a la programación en lenguajes estructurados y sirve tanto para cursos introductorios de carácter semestral, tales como *Introducción a la Programación, Metodología de la Programación* o *Fundamentos de programación,* en primeros cursos de carreras de ingeniería informática, ingeniería de sistemas y licenciatura en informática o en sistemas de información, y asignaturas de programación de ingeniería y de ciencias. Contiene esta parte los fundamentos teóricos y prácticos relativos a la or-

ganización de una computadora y los lenguajes de programación, así como la descripción de las herramientas de programación más frecuentemente utilizadas en el campo de la programación. Se incluyen también en esta parte los elementos básicos constitutivos de un programa y las herramientas de programación utilizadas, tales como algoritmos, diagramas de flujo, etc. La segunda mitad de esta primera parte es una descripción teórico-práctica de las estructuras utilizadas para controlar el flujo de instrucciones de un programa y una descripción detallada del importante concepto de subprograma (procedimiento/función), piedra angular de la programación modular y estructurada.

Capítulo 1. ***Computadoras y lenguajes de programación.*** Las computadoras son herramientas esenciales en muchas áreas de la vida: profesional, industrial, empresarial, académica, ... en realidad, en casi todos los campos de la sociedad. Las computadoras funcionan correctamente con la ayuda de los programas. Los programas se escriben mediante lenguajes de programación que previamente se han escrito en algoritmos u otras herramientas, tales como diagramas de flujo. Este capítulo introductorio describe la organización de una computadora y sus diferentes partes junto con el concepto de programa y de lenguaje de programación. Así mismo y al objeto de que el lector pueda entender los fundamentos teóricos en que se asienta la programación, se incluye una breve historia de los lenguajes de programación más influyentes y que, en el caso de la tercera edición, han servido de inspiración para la nueva versión del pseudocódigo **UPSAM 2.0**: es decir, C, C++, Java y C#, con referencias lógicas al histórico Pascal.

Capítulo 2. ***Resolución de problemas con computadora y herramientas de programación.*** En este capítulo se describe métodos para la resolución de problemas con computadora y con un lenguaje de programación (en nuestro caso el pseudolenguaje o lenguaje algorítmico UPSAM 2.0). Se explican las fases de la resolución de un problema junto con las técnicas de programación modular y estructurada. Se inicia en este capítulo la descripción del concepto, función y uso de algoritmo. Uno de los objetivos más importantes de este libro es el aprendizaje, diseño y construcción de algoritmos.

Capítulo 3. ***Estructura general de un programa.*** Enseña la organización y estructura general de un programa así como su creación y proceso de ejecución. Se describen los elementos básicos de un programa: tipos de datos, constantes, variables y entradas/salidas de datos. También se introduce al lector en la operación de asignación así como en el concepto de función interna. De igual forma se estudia los importantes conceptos de expresiones y operaciones junto con sus diferentes tipos.

Capítulo 4. ***Flujo de control I: Estructuras selectivas.*** Introduce al concepto de estructura de control y, en particular, estructuras de selección, tales como **`si-entonces`** `("if-then")`, **`según_sea/caso_de ("switch/case")`**. Se describen también las estructuras de decisión anidadas. Así mismo se explica también la «denostada» sentencia **`ir_a (goto)`**, cuyo uso no se recomienda pero sí el conocimiento de su funcionamiento,

Capítulo 5. ***Flujo de control II: Estructuras repetitivas.*** El capítulo introduce las estructuras repetitivas **`(mientras ("while"), hacer-mientras ("do-while"), repetir ("repeat"), desde/para ("for")`**. Examina la repetición *(iteración)* de sentencias en detalle y compara los bucles controlados por centinela, bandera, etc. Explica precauciones y reglas de uso de diseño de bucles. Compara los tres diferentes tipos de bucles, así como el concepto de bucles anidados.

Capítulo 6. ***Subprogramas.*** La resolución de problemas complejos se facilita considerablemente si se divide en problemas más pequeños (subproblemas). La resolución de estos problemas se realiza con *subalgoritmos* (subprogramas) que a su vez se dividen en dos grandes categorías: *funciones* y *procedimientos.*

PARTE II. ESTRUCTURA DE DATOS

Esta parte es clave en el aprendizaje de técnicas de programación. Tal es su importancia que los planes de estudio de cualquier carrera de ingeniería informática o de ciencias de la computación incluye una asignatura troncal denominada *Estructura de datos.*

Capítulo 7. ***Estructuras de datos I (arrays, estructuras y registros).*** Examina la estructuración de los datos en *arrays* o grupos de elementos dato del mismo tipo. El capítulo presenta numerosos ejemplos de *arrays* de uno, dos o múltiples índices. También se explican los otros tipos de estructuras de datos básicas: estructuras y registros. Estas estructuras de datos permiten encapsular en un tipo de dato definido por el usuario otros datos heterogéneos. Asi mismo se describe el concepto de arrays de estructuras y arrays de registros.

Capítulo 8. ***Cadenas.*** Se examina el concepto de carácter y de cadena (`String`) junto con su declaración e inicialización. Se introducen conceptos básicos de manipulación de cadenas:lectura y asignación junto con operaciones básicas, tales como longitud, concatenación, comparación, conversión y búsqueda de caracteres y cadenas. Las operaciones de tratamiento de caracteres y cadenas son operaciones muy usuales en todo tipo de programas.

Capítulo 9. ***Archivos (Ficheros).*** El concepto de archivo junto con su definición e implementación es motivo de estudio en este capítulo. Los tipos de archivos más usuales junto con las operaciones básicas de manipulación se estudian con detenimiento.

Capítulo 10. ***Ordenación, búsqueda e intercalación.*** Las computadoras emplean una gran parte de su tiempo en operaciones de búsqueda, clasificación y mezcla de datos. Los archivos se sitúan adecuadamente en dispositivos de almacenamiento externo que son más lentos que la memoria central pero que, por el contrario, tienen la ventaja de almacenamiento permanente después de apagar la computadora. Se describen los algoritmos de los métodos más utilizados en el diseño e implementación de programas.

Capítulo 11. ***Ordenación, Búsqueda y Fusión externa (Archivos).*** Normalmente los datos almacenados de modo permanente en dispositivos externos requieren para su procesamiento el almacenamiento en la memoria central. Por esta circunstancia, las técnicas de ordenación y búsqueda sobre arrays y vectores comentados en el capítulo anterior necesitan una profundización en cuanto a técnicas y métodos. Una de las técnicas más importantes es la fusión o mezcla. En el capítulo se describen las técnicas de manipulación externa de archivos.

Capítulo 12. ***Estructuras de datos lineales (pilas, colas y listas enlazadas).*** Una lista enlazada es una estructura de datos que mantiene una colección de elementos, pero el número de ellos no se conoce por anticipado o varía en un amplio rango. La lista enlazada se compone de elementos que contienen un valor y un puntero. El capítulo describe los fundamentos teóricos y las operaciones que se pueden realizar en la lista enlazada. También se describen los distintos tipos de listas enlazadas, tales como doblemente enlazadas y circulares. Las ideas abstractas de pila y cola se describen en el capítulo. Pilas y colas se pueden implementar de diferentes maneras, bien con vectores (arrays) o con listas enlazadas.

Capítulo 13. ***Estructuras de datos no lineales (árboles y grafos).*** Los árboles son otro tipo de estructura de datos dinámica y no lineal. Se estudian las operaciones básicas en los árboles junto con sus operaciones fundamentales.

Capítulo 14. ***Recursividad.*** El importante concepto de recursividad (propiedad de una función de llamarse a sí misma) se introduce en el capítulo junto con algoritmos complejos de ordenación y búsqueda en los que además se estudia su eficiencia.

III. PROGRAMACIÓN ORIENTADA A OBJETOS

Nuestra experiencia en la enseñanza de la programación orientada a objetos a estudiantes universitarios data de finales de la década de los ochenta. En este largo periodo, los primitivos y básicos conceptos de orientación a objetos se siguen manteniendo desde el punto de vista conceptual y práctico, tal y como se definieron hace treinta años. Hoy la programación orientada a objetos es una clara rea-

lidad y por ello cualquier curso de introducción a la programación aconseja, al menos, incluir un pequeño curso de orientación a objetos que puede impartirse como un curso independiente, como complemento de la parte II o como parte de un curso completo de introducción a la programación que comienza en el Capítulo 1.

Capítulo 15. ***Conceptos fundamentales de orientación a objetos.*** Este capítulo describe los conceptos fundamentales de la orientación a objetos: clases, objetos y herencia. La definición y declaración de una clase junto con su organización y estructura se explican detenidamente. Se describen también otros conceptos importantes, tales como polimorfismo, ligadura dinámica y sobrecarga y un resumen de la terminología orientada a objetos.

Capítulo 16. ***Diseño de clases y objetos: representaciones gráficas en UML.*** Una de las tareas fundamentales de un programador es el diseño y posterior implementación de una clase y de un objeto en un lenguaje de programación. Para realizar esta tarea con eficiencia se exige el uso de una herramienta gráfica. UML es el lenguaje de modelado unificado estándar en el campo de la ingeniería de software y en el capítulo se describen las notaciones gráficas básicas de clases y objetos.

Capítulo 17. ***Relaciones: Asociación, Generalización y Herencia.*** En este capítulo se introducen los conceptos fundamentales de las relaciones entre clases: asociación, agregación y generalización/especialización. Se describen todas estas relaciones así como las notaciones gráficas que las representan en el lenguaje de modelado UML.

PARTE IV. METODOLOGÍA DE LA PROGRAMACIÓN Y DESARROLLO DE SOFTWARE

En esta parte se describen reglas prácticas para la resolución de problemas mediante programación y a su posterior desarrollo de software. Estas reglas buscan proporcionar al lector reglas de puesta a punto de programas junto con directrices de metodología de programación que faciliten al lector la tarea de diseñar y construir programas con calidad y eficiencia junto con una introducción a la ingeniería de software.

Capítulo 18. ***Resolución de problemas y desarrollo de software: Metodología de la programación.*** En el capítulo se analiza el desarrollo de un programa y sus diferentes fases: análisis, diseño, codificación, depuración, pruebas y mantenimiento. Estos principios básicos configuran la ingeniería de software como ciencia que pretende la concepción, diseño y construcción de programas eficientes.

APÉNDICES

En todos los libros dedicados a la enseñanza y aprendizaje de técnicas de programación es frecuente incluir apéndices de temas complementarios a los explicados en los capítulos anteriores. Estos apéndices sirven de guía y referencia de elementos importantes del lenguaje y de la programación de computadoras.

Apéndice A. ***Especificaciones del lenguaje algorítmico UPSAM 2.0.*** Se describen los elementos básicos del lenguaje algorítmico en su versión 2.0 junto con la sintaxis de todos los componentes de un programa. Así mismo se especifican las palabras reservadas y símbolos reservados. En relación con la versión 1.0 de UPSAM es de destacar una nueva guía de especificaciones de programación orientada a objetos, novedad en esta versión y que permitirá la traducción del pseudocódigo a los lenguajes orientados a objetos tales como C++, Java o C#.

Apéndice B. ***Prioridad de operadores.*** Tabla que contiene todos los operadores y el orden de prioridad y asociatividad en las operaciones cuando aparecen en expresiones.

Apéndice C. ***Guía de sintaxis de Pascal y Turbo Pascal.*** Aunque ya está muy en desuso en la enseñanza de la programación el lenguaje Pascal, su sintaxis y estructura sigue siendo un modelo excelente de aprendizaje y es esta la razón de seguir incluyendo esta guía de sintaxis en la obra.

Apéndice D. ***Guía de sintaxis del lenguaje ANSI C.*** Especificaciones, normas de uso y reglas de sintaxis del lenguaje de programación C en su versión estándar ANSI/ISO.

Apéndice E. ***Guía de sintaxis del lenguaje ANSI/ISO C++ estándar.*** Especificaciones, normas de uso y reglas de sintaxis del lenguaje de programación C++ en su versión estándar ANSI/ISO.

Apéndice F. ***Guía de sintaxis del lenguaje Java 2.*** Descripción detallada de los elementos fundamentales del estándar Java: especificaciones, reglas de uso y de sintaxis.

Apéndice G. ***Guía de sintaxis del lenguaje C#.*** Descripción detallada de los elementos fundamentales del lenguaje C#: especificaciones, reglas de uso y de sintaxis.

Apéndice H. ***Palabras reservadas: C++, Java y C#.*** Listados de palabras reservadas (clave) de los lenguajes de programación C++, Java y C#. Se incluye también una tabla comparativa de las palabras reservadas de los tres lenguajes de programación.

Apéndice I. ***Glosario de palabras reservadas de C/C++.*** Glosario terminológico de las palabras reservadas del lenguaje de programación C/C++ con una breve descripción y algunos ejemplos de uso de cada palabra.

Apéndice I. ***Glosario de palabras reservadas de C#.*** Glosario terminológico de las palabras reservadas del lenguaje de programación C# con una breve descripción y algunos ejemplos de uso de cada palabra.

Apéndice K. ***Código ASCII y Unicode.*** Tablas de los códigos de caracteres que se utilizan en programas de computadoras. El código ASCII es el más universal y empleado de modo masivo por programadores de todo el mundo y, naturalmente, es el que utilizan la mayoría de las computadoras actuales. Unicode es un lenguaje mucho más amplio que utilizan las computadoras personales para realizar programas y aplicaciones en cualquier tipo de computadora y en Internet. Unicode proporciona un número único para cada carácter, sin importar la plataforma, sin importar el programa, sin importar el idioma. La importancia de Unicode reside, entre otras cosas, en que está avalado por líderes de la industria tales como Apple, HP, IBM. Microsoft, Oracle, Sun, entre otros. También es un requisito para los estándares modernos, tales como XML, Java, C#, etc.

Apéndice J. ***Recursos de programación: Libros, Revistas, Web, Lecturas recomendadas.*** Enumeración de los libros más sobresalientes empleados por el autor en la escritura de esta obra, así como otras obras importantes de referencia que ayuden al lector que desee profundizar o ampliar aquellos conceptos que considere necesario conocer con más detenimiento. Listado de sitios Web de interés para la formación en Java, tanto profesionales como medios de comunicación, especialmente revistas especializadas.

AGRADECIMIENTOS

Un libro nunca es fruto único del autor, sobre todo si el libro está concebido como libro de texto y autoaprendizaje, y pretende llegar a lectores y estudiantes de informática y de computación, y, en general, de ciencias e ingeniería, así como autodidactas en asignaturas tales como programación (introducción, fundamentos, avanzada, etc.). Esta obra no es una excepción a la regla y son muchas las personas que nos han ayudado a terminarla. En primer lugar deseo agradecer a mis colegas de la Universidad Pontificia de Salamanca en el campus de Madrid, y en particular del Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software de la misma que desde hace muchos años nos ayudan y colaboran en la impartición de las diferentes asignaturas del departamento y sobre todo en

la elaboración de los programas y planes de estudio de las mismas. A todos ellos les agradezco públicamente su apoyo y ayuda.

Además de todos los profesores citados en el prólogo de la primera edición que han seguido contribuyendo con sus aportaciones al nacimiento de la versión 2.0 en sus respectivas clases, queremos añadir los nuevos profesores que imparten asignaturas relacionadas con programación y que siguen el lenguaje algorítmico en sus clases y nos han proporcionado críticas, comentarios, reflexiones, sugerencias y consejos para la mejora del lenguaje. Estos profesores, y con el riesgo de olvidar algún nombre de algún colega y amigo —a quien pedimos disculpas por anticipado si ese hecho se produjera—, destacamos a los siguientes:

- Daniel Zapico
- Elisa García
- Francisco Jiménez
- Franciso Pérez Sorrosal
- Gloria García
- Héctor Castán
- María del Pilar Dorrego
- Marta Fernández
- Mónica Vázquez
- Oscar Sanjuán
- Salvador Sánchez
- Virginia Sierra
- Cristina Sequeira

Esta tercera edición ha sido leída y revisada con toda minuciosidad y —fundamentalmente— rigurosidad por los siguientes profesores de la Universidad Pontificia de Salamanca en el campus de Madrid: Luis Rodríguez Baena, Matilde Fernández Azuela, Víctor Martín García, Lucas Sánchez e Ignacio Zahomero Martínez (mi eterno agradecimiento).

Muchos otros profesores españoles y latinoamericanos me han ayudado en la concepción y realización de esta obra, de una u otra manera, apoyándome con su continua colaboración y sugerencia de ideas la puesta en marcha de asignaturas del área de programación, en temas tan variados como *Fundamentos de Programación, Lenguajes de programación tales como BASIC, Visual Basic, Pascal, C,C++, Java* o *Delphi*. Citar a todos ellos me llevaría páginas completas. Si al menos y como reconocimiento silencioso, decir que además de España, se incluyen todos los países latinoamericanos desde Méjico a Argentina y Chile en el cono sur, pasando por Centroamérica, Guatemala o República Dominicana. En cualquier forma, sí quería destacar de modo especial a la profesora M.ª Eugenia Vallesany de la Universidad Nacional del Nordeste de Corrientes en Argentina, por la labor de rigurosa revisión que ha realizado sobre la segunda edición y la gran cantidad de sugerencias y propuestas que me ha hecho para esta nueva edición motivada fundamentalmente por su extraordinaria y valiosa investigación al mundo de los algoritmos y de la programación. **A todos ellos y a todos nuestros lectores y alumnos de España y Latinoamérica, nuestro agradecimiento eterno.**

Además de a nuestros compañeros en la docencia y a nuestros alumnos, no puedo dejar de agradecer, una vez más, a mi editora —y sin embargo amiga— Concha Fernández, las constantes muestras de afecto y comprensión que siempre tiene con mis obras. Esta ocasión, como no era menos, tampoco ha sido una excepción. Sin embargo. ahora he de resaltar esa gran amistad que nos une. La elaboración de esta obra por mil circunstancias ha entrañado, más que nunca, tal vez muchas más di-

ficultades que otras obras nuestras. De nuevo y con gran paciencia, nos ha ayudado, comprendido y tolerado nuestras mil y una duda, sugerencias, retrasos, etc. No podemos menos de expresar nuestro infinito reconocimiento y agradecimiento. Sin esta comprensión y su apoyo continuo posiblemente hoy todavía no habría visto la luz esta obra, debido a mis grandes retrasos en la entrega del original y posteriores revisiones de imprenta. Con todo nuestro corazón en la mano, nuestro eterno agradecimiento. Pero en esta ocasión también deseo agradecer las muchas atenciones que Mely de McGraw-Hill ha puesto en la edición de esta obra. Sus consejos, ideas y sugerencias, unido a su gran paciencia y comprensión con el autor por sus muchos retrasos en la entrega de originales, han hecho que la obra haya sido mejorada considerablemente en el proceso de edición.

Naturalmente —y aunque ya los hemos citado anteriormente—, no podemos dejar de agradecer a nuestros numerosos alumnos, estudiantes y lectores, en general, españoles y latinoamericanos, que continuamente nos aconsejan, critican y nos proporcionan ideas para mejoras continuas de nuestros libros. Sin todo lo que hemos aprendido, seguimos aprendiendo y seguiremos aprendiendo de ellos y sin su aliento continuo me sería prácticamente imposible terminar mis nuevas obras y, en especial, este libro. De modo muy especial deseo reiterar mi agradecimiento a tantos y tantos colegas de universidades españolas y latinoamericanas que apoyan nuestra labor docente y editorial. Mi más sincero reconocimiento y agradecimiento, una vez más, a todos: alumnos, lectores, colegas, profesores, maestros, monitores y editores. Muy bien sé que siempre estaremos en deuda con vosotros. Mi único consuelo es que vuestro apoyo me sigue dando fuerza en esta labor académica y que allá por donde mis derroteros profesionales me llevan siempre está presente ese inmenso e impagable agradecimiento a esa enorme ayuda que me prestáis. **Gracias, una vez más, por vuestra ayuda.**

En Carchelejo (Jaén) y en Madrid, Navidades del 2002.

El autor

PRÓLOGO DE LA SEGUNDA EDICIÓN

El primer curso de programación de un currículum de Ciencias e Ingeniería es probablemente uno de los más importantes. Para la mayoría de los estudiantes, el conocimiento inicial eficiente de conceptos básicos tales como algoritmos y estructura de datos, así como el modo de aprender a resolver problemas mediante computadoras, suele ser vital debido a la trascendencia que un aprendizaje gradual y correcto supondrá para su carrera y sobre todo para sus primeros cursos en la universidad (facultades, escuelas o institutos tecnológicos en Latinoamérica). Por esta causa fundamental la mayoría de los *currícula* incluyen asignaturas de carácter troncal u obligatorio, tales como ***Metodología de la Programación, Introducción a la Programación*** o ***Fundamentos de Programación,*** cuyo objetivo es enseñar a los alumnos los conceptos fundamentales de la programación, especialmente la *programación estructurada,* que les permita adquirir una orientación específica, básica y fundamental y que les garantice afrontar con éxito el estudio de las restantes asignaturas del *currículum,* tales como Tecnología de la Programación, Sistemas Operativos, Programación Avanzada, Estructuras de Datos, etc., o puramente científica, como Cálculo Numérico, Algoritmia Avanzada, etc.

Para tratar de conseguir los objetivos citados, decidimos escribir y publicamos en 1988 la primera edición del libro ***Fundamentos de programación. Algoritmos y estructura de datos,*** que fue la continuación de uno anterior —***Metodología de la programación***— que se publicó en 1986 y que se centró fundamentalmente en el diseño básico de algoritmos. Así en esta obra nos centramos en los conceptos y diseño de algoritmos junto con los fundamentos teórico-prácticos de estructuras de datos y de programación estructurada; por esta razón nos apoyamos en herramientas clásicas de programación, tales como diagramas de flujo, diagramas N-S y sobre todo pseudocódigo. En aquellos años el uso del pseudocódigo en inglés estaba extendido, pero no así el pseudocódigo en español, por lo que pensamos que sería muy interesante el empleo del español como soporte del pseudocódigo, de modo que los lectores de nuestro idioma pudieran introducirse de un modo más fácil en el aprendizaje de la programación. Desde entonces, no sólo nuestras obras sino muchas otras han comenzado a utilizar el pseudocódigo en español y, tanto en España como en Latinoamérica, ha pasado a ser una herramienta decisiva empleada no sólo en los cursos de introducción a la programación sino en cursos avanzados y en otros similares.

Estos hechos, junto con las numerosas sugerencias, recomendaciones, consejos y críticas de colegas y alumnos españoles y latinoamericanos. nos ha llevado a escribir la segunda edición de este libro, apoyándonos fundamentalmente en el pseudocódigo como lenguaje algorítmico por excelencia.

En estos ocho años transcurridos desde la primera edición, el libro ha sido utilizado en numerosas universidades, escuelas de ingenieros e institutos tecnológicos, a la vez que la disciplina informática —en general— y la programación —en particular— han continuado evolucionando y madurando. Nuevos conceptos y nuevos lenguajes de programación han conducido a nuevas técnicas de programación

y al diseño de nuevos *currícula* en carreras de Ingeniería Informática y en Sistemas Computacionales así como en Licenciatura y Diplomatura e Ingeniería Técnica Informática. A mediados de los años ochenta tres lenguajes de programación predominaban en los ambientes informáticos: BASIC, Pascal y COBOL, junto con el clásico FORTRAN y los lenguajes emergentes C, Modula-2 y Ada. En la actualidad, Pascal y COBOL siguen siendo clásicos como primer lenguaje de programación y como lenguaje profesional, por excelencia, manteniéndose el clásico FORTRAN para aplicaciones científicas, sobre todo ya desarrolladas, y están implantándose cada vez con mayor intensidad lenguajes tales como C/C++ y Ada-95, comenzando a adquirir una notable influencia otros lenguajes como Visual BASIC o Delphi.

El uso de un lenguaje de programación, claramente, es una parte importante de cualquier curso de introducción a la programación; pero este hecho no debe oscurecer otros conceptos importantes. Así, este libro se basa en la premisa de que el estudiante/lector debe aprender a programar con la filosofía de usar la sintaxis y estructura de un lenguaje de programación pero no de *un lenguaje de programación específico.* Por esta causa, y teniendo presente la experiencia con gran número de alumnos de nuestra universidad y de las muchas otras universidades donde se ha utilizado la primera edición como texto básico, hemos decidido utilizar como lenguaje de programación algorítmico el pseudocódigo en español o castellano **UPSAM** (Universidad Pontificia de Salamanca en Madrid) fruto de la experiencia acumulada en el Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software de la Facultad y Escuela Universitaria de Informática de la Universidad Pontificia de Salamanca en Madrid, durante los diez últimos años.

EL LENGUAJE ALGORÍTMICO DE PROGRAMACIÓN UPSAM

El lenguaje algorítmico **UPSAM** o pseudocódigo español utilizado como referencia básica en el texto ha sido creado, diseñado y experimentado por todos los profesores que han impartido las asignaturas de ***Fundamentos de Programación*** en las carreras universitarias *Técnicos en Informática Fundamental (TIF), Diplomados en Informática* y *Licenciados en Informática,* y ***Metodología de la Programación*** en los cursos de postgrado *Master, Técnicos* y *Expertos en Informática* de la Universidad Pontificia de Salamanca en Madrid y en su Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software.

La guía de sintaxis del lenguaje algorítmico **UPSAM** se describe en el Apéndice A. Ha sido redactada por los profesores Matilde Fernández, Luis Rodríguez y Luis Joyanes, y revisada y corregida por los profesores Luis Villar, Joaquín Abeger, Ángel Hermoso, Antonio Muñoz, Lucas Sánchez, Víctor Martín y Rafael Ojeda, que han aportado sugerencias y mejoras al borrador original. Asimismo también han contribuido a la redacción final con sugerencias y aportaciones y pertenecen al equipo creador del lenguaje **UPSAM** los siguientes profesores de la citada Universidad:

- Ángela Carrasco
- Francisco Oliva
- Isabel Torralvo
- M.ª del Mar García
- Mercedes Vargas
- Miguel Sánchez

Profesores de asignaturas tales como Programación I y II, Estructura de Datos, Programación de Sistemas, Metodología de la Programación, Programación de Gestión, Sistemas Operativos y otras han utilizado el pseudocódigo UPSAM en una u otra forma y han dado sugerencias también que se han incorporado a la redacción final. Así, y con el riesgo de olvidar algún nombre de algún colega y amigo —a quien pedimos disculpas por anticipado—, destacamos a:

- Amalia Oñate
- Andrés Castillo
- Ángel González
- Antonio Reus
- Daniel García
- David Juanes
- Enrique Torres
- Fernando Davara
- Francisco Agüera
- Francisco Mata
- Jesús Pérez
- José González
- José Luis Andrés
- Juan Ignacio Pérez
- Juan Manuel Dodero
- Juan Sanz
- Julián Margolles
- Luis Doreste
- M.ª Luisa Díez
- Manuel Redón
- Manuel Riquelme
- Paloma Centenera
- Rosa Hernández
- Virgilio Yagüe

PROPÓSITO DEL LIBRO

El propósito principal de este libro es servir como material de lectura y estudio para un curso de introducción a la programación en una carrera universitaria de Ciencias e Ingeniería, preferentemente *Ciencias de la Computación, Ingeniería Informática o en Sistemas Computacionales, Diplomatura y Licenciatura en Informática.*

El libro sigue las directrices fundamentales de asignaturas troncales de los planes de estudio españoles ***Introducción a la programación y Metodología de la Programación,*** así como los temas principales de los cursos **CS1** y **CS2** de la **ACM** (Association for Computing Machinery).

Se siguen las reglas clásicas de construcción de diseño de programas estructurados, con énfasis especial en algoritmos y estructuras de datos. El libro ha conservado la organización de la primera edición, incluyendo gran cantidad de ejemplos y ejercicios resueltos con el objeto de que el alumno pueda comprobar su progreso. Incluye además en cada capítulo Actividades de Programación que contiene numerosos ejercicios y problemas resueltos de complejidad creciente.

La actualización y puesta al día de los nuevos lenguajes de programación nos ha hecho incluir guías de sintaxis de los lenguajes Pascal, Turbo Pascal, C y C++, así como reglas de conversión del lenguaje algorítmico a los citados lenguajes, de modo que el lector pueda utilizar a la terminación de la lectura del libro o en paralelo alguno de los lenguajes anteriores y construir programas en los mismos sin ninguna dificultad.

Así, se han cuidado especialmente los siguientes tópicos:

1. Escribir código modular.
2. Utilizar abstracción en el proceso de desarrollo de software.
3. Escritura de documentación del código fuente.
4. Legibilidad y claridad en el código fuente.
5. Semántica consistente para todos los procesos de un programa.
6. Utilizar estrictamente las reglas de la programación estructurada.

LIBRO COMPLEMENTARIO

Todos los cursos de programación deben apoyarse en la resolución de gran número de problemas que permitan al alumno/lector adquirir práctica que le faciliten el aprendizaje. Por esta causa se han incluido gran cantidad de ejercicios, problemas y actividades de programación en el texto. Además, nuestra experiencia docente y la petición de numerosos alumnos y colegas nos llevó a considerar escribir un libro de problemas que resolviera un buen número de ejercicios propuestos y otros nuevos, de modo que el alumno pudiese comprobar sus avances no sólo a través de libro con actividades de programación, sino con ayuda de un libro clásico de problemas pero con la aportación y experiencia metodológica de nuestra universidad. Por esta causa mis colegas Matilde Fernández, Luis Rodríguez y yo mismo hemos escrito el libro ***Fundamentos de Programación. Libro de problemas***, también publicado en McGraw-Hill, que puede ser utilizado de modo complementario a este libro o de modo totalmente independiente, ya que ha sido diseñado explícitamente para cumplir ambos requisitos y especialmente para facilitar el aprendizaje de la programación.

ORGANIZACIÓN DEL LIBRO

El libro se ha estructurado en catorce capítulos y ocho apéndices con una organización similar en todos ellos que incluyen desde conceptos teóricos a reglas y conceptos prácticos junto con actividades de programación y ejercicios propuestos al lector para que tras su realización éste pueda comprobar y medir su progreso en el mundo de la programación.

El Capítulo 1 presenta una breve descripción del concepto de sistema de procesamiento de información. Introduce al concepto de algoritmo y las primeras construcciones y elementos de un lenguaje de programación. Se describen los tipos de datos y las operaciones básicas que se realizan en cualquier programa.

El Capítulo 2 analiza los métodos para resolver problemas con computadoras y las herramientas de programación necesarias para ello. Se describen las etapas clásicas utilizadas en la resolución de problemas así como las herramientas clásicas utilizadas para su descripción, tales cómo pseudocódigos, diagramas de flujo y diagramas N-S.

En el Capítulo 3 se describe el concepto de programa y sus elementos básicos, junto con las instrucciones y tipos de instrucciones más usuales. Se introduce el concepto de *flujo de control* y se incluyen los primeros problemas planteados de cierta complejidad y su resolución con alguna de las herramientas descritas en el Capítulo 2.

La programación estructurada y sus técnicas es uno de los conceptos claves que se describen y utilizan a lo largo del texto. El Capítulo 4 explica las técnicas de programación empleadas en la progra-

mación modular y estructurada; se amplía el concepto de flujo de control desarrollado en el Capítulo 3 y se describen con una gran cantidad de ejemplos y ejercicios aclaratorios las estructuras clásicas empleadas en la programación estructurada: secuenciales, selectivas y repetitivas.

El Capítulo 5 trata de los subprogramas (subalgoritmos): procedimientos y funciones. Los temas descritos incluyen la correspondencia de argumentos y parámetros, así como la comunicación entre el programa principal y los subprogramas y entre ellos entre sí. El método de invocación de un subprograma junto con los valores devueltos en el caso de las funciones son otro de los temas considerados en el capítulo. Los problemas planteados por el uso inadecuado de variables globales conocidos como *efectos laterales* se analizan también. El concepto de *recursión* o *recursividad* se introduce en el capítulo como técnica fundamental de diseño del algunos problemas de naturaleza recursiva.

El concepto de **array** (*tabla, matriz* o *vector,* **«arreglo»** en Latinoamérica) es el tema central del Capítulo 6. El proceso de arrays de una dimensión (vectores) se examina en primer lugar. Se describen algunas aplicaciones típicas con vectores. A continuación se analizan los arrays multidimensionales, con especial hincapié en los arrays bidimensionales o tablas.

El procesamiento de datos tipo carácter y las cadenas de caracteres se describen en el Capítulo 7. Algunas operaciones básicas con cadenas junto a funciones intrínsecas para tratamiento de cadenas se analizan también en este capítulo.

El Capítulo 8 describe el concepto de archivo externo, su organización, almacenamiento y recuperación en dispositivos externos tales como discos y cintas. Los diferentes tipos de archivos y las operaciones típicas que se realizan con ellos son tema central del capítulo. Las organizaciones secuenciales, aleatorias o de acceso directo y las indexadas se analizan y describen junto con los algoritmos de manipulación de los archivos.

Las aplicaciones más importantes y frecuentes en cualquier algoritmo son la ordenación, búsqueda y mezcla o intercalación de información. Los algoritmos más populares y reconocidos relativos a arrays unidimensionales se analizan en el Capítulo 9. Estos algoritmos se consideran internos, dado que el procesamiento de los mismos se realiza en memoria central, para diferenciarlos de la manipulación de los archivos que se consideran externos y que se almacenan en soportes de almacenamiento externo. La ordenación, búsqueda y fusión externa se describen en el Capítulo 10, mientras que las operaciones internas correspondientes se describen en el Capítulo 9.

El Capítulo 11 ofrece una introducción al estudio de las estructuras de datos lineales. Estructuras de datos simples, tales como listas enlazadas, pilas y colas, se examinan a lo largo del capítulo. Se describen un gran número de operaciones importantes y sus implementaciones mediante arrays y punteros o apuntadores.

El Capítulo 12 estudia las estructuras de datos no lineales: árboles y grafos. El tipo de árbol más usual es el árbol binario y se describe en profundidad; de igual forma se describen los grafos. La terminología usual empleada en los árboles y grafos junto con algoritmos sencillos se describen en el capítulo.

En el Capítulo 13 se dan reglas para realizar un diseño eficaz y eficiente de programas. El estilo de programación es sin duda tema clave en la construcción moderna de programas. El elevado tamaño de la memoria central de las computadoras junto con las grandes capacidades de almacenamiento de los discos externos de las mismas han conducido a considerar el estilo de programación como uno de los factores clave en el desarrollo de programas. Las reglas para obtener un programa de calidad se analizan a lo largo del capítulo.

Una de las herramientas clásicas empleadas en el análisis de un problema es la tabla de decisión. En el Capítulo 14 se analiza el concepto, tipos, construcción y conversión de tablas de decisión en programas. La tabla de decisión es una herramienta de gestión eficaz que se complementa excelentemente con el diseño de algoritmos para la resolución de problemas, especialmente del campo de la gestión.

El Apéndice A describe las especificaciones del lenguaje algorítmico **UPSAM** *(Universidad Pontificia de Salamanca en Madrid)* cuyo soporte fundamental es el pseudocódigo y que ha sido utilizado en todo el texto. El Apéndice B es una guía de usuario del lenguaje Pascal estándar tal como fue concebido por su inventor Niklaus Wirth: es el lenguaje clave utilizado en la mayoría de las universidades de todo el mundo para enseñar el aprendizaje de la programación a los alumnos de los prime-

ros cursos. Hoy día no se utiliza con profusión pero su conocimiento es esencial para cualquier persona que desee aprender técnicas de programación estructurada. El Apéndice C es otra guía de usuario o de sintaxis de Turbo/Borland Pascal, la versión más popular del lenguaje implantada en computadoras personales tipo PC o Macintosh y bajo sistemas operativos tales como DOS, Windows 3.1 o Windows 95.

Los Apéndices D y E se dedican a describir los lenguajes de programación más utilizados en la actualidad en el mundo profesional: C y C++ (este último como base de la programación orientada a objetos). La estructura del lenguaje C ANSI desarrollado por Kernighan y Ritchie se describe en el Apéndice D, mientras que en el Apéndice E se describen las diferencias entre C y C++ junto con las características específicas de C++.

El Apéndice F muestra reglas para convertir algoritmos escritos en pseudecódigo o mejor en lenguaje UPSAM a lenguajes de programación clásicos tales como Pascal, FORTRAN y Modula-2. Los Apéndices A a E deben permitir al alumno la conversión de cualquier programa a los lenguajes Pascal/Turbo Pascal o C/C++.

El Apéndice G es un breve glosario de términos de programación utilizados a lo largo del libro y *que facilitarán al lector la introducción a la terminología clásica de programación.* El Apéndice H incluye tablas con los códigos de caracteres estándar ASCII, empleados por la mayoría de las computadoras fabricadas en la actualidad. El libro termina con una amplia bibliografía relativa a técnicas de programación, programación estructurada y los lenguajes de programación citados en el texto, tales como Pascal, COBOL, C, C++, Ada, etc.

AGRADECIMIENTOS

Los profesores citados con anterioridad han contribuido de una u otra forma a la creación del lenguaje algorítmico UPSAM y, por consiguiente, debemos agradecerles de nuevo su contribución al mismo, así como numerosas sugerencias y consejos para mejorar la primera edición. Sin embargo, en esta segunda edición nos han prestado una especial contribución, en forma de revisión y corrección de la primera edición o bien de esta segunda edición, una serie de profesores a los que deseamos expresar nuestro reconocimiento y agradecimiento especial. Estos profesores son: Antonio Muñoz, Ángel Hermoso, Luis Villar, Luis Rodríguez y Matilde Fernández, pertenecientes al Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software, de la Facultad y Escuela Universitaria de Informática de la Universidad Pontificia de Salamanca en Madrid.

Numerosos profesores de universidades e institutos tecnológicos de Latinoamérica nos han sugerido también ideas, aportaciones y sugerencias que hemos tenido en cuenta en esta segunda edición. Deseamos destacar de modo especial la revisión y corrección completa de la primera edición realizada por Óscar Manuel Arango Jaramillo, profesor titular de Universidad Eant del Departamento de Ingeniería de Sistemas de Medellín (Colombia).

Por último, considerar nuestro agradecimiento más especial a los numerosos alumnos de España y Latinoamérica que han estudiado con este libro y se han iniciado en la programación con él; muchas de sus críticas, sugerencias y propuestas han sido un gran acicate para que esta segunda edición pudiese ver la luz. Mi reconocimiento más sincero es para todos estos lectores y alumnos cuyo aliento y ánimo nunca me han faltado.

Majadahonda (Madrid), verano de 1996
El autor

PARTE I

ALGORITMOS Y HERRAMIENTAS DE PROGRAMACIÓN

1

COMPUTADORAS Y LENGUAJES DE PROGRAMACIÓN

CONTENIDO

1.1. Organización de una computadora.
1.2. *Hardware*.
1.3. Dispositivos de almacenamiento de información.
1.4. La computadora personal ideal para programación.
1.5. El *software* (los programas).
1.6. Los lenguajes de programación.
1.7. Traductores de lenguaje.
1.8. Historia de los lenguajes de programación.
ACTIVIDADES DE PROGRAMACIÓN RESUELTAS.
REVISIÓN DEL CAPÍTULO.
Conceptos clave.
Resumen.
EJERCICIOS.

Las computadoras (ordenadores) son herramientas esenciales en muchas áreas: industria, gobierno, ciencia, educación..., en realidad en casi todos los campos de nuestras vidas. El papel (rol) de los programas de computadoras es esencial; sin una lista de instrucciones a seguir, la computadora es virtualmente inútil. Los lenguajes de programación nos permiten escribir esos programas y, por consiguiente, comunicarnos con las computadoras.

Una computadora procesa datos y los convierte en información significativa. Para conseguir esos resultados, un programador necesita conocimientos tanto de *hardware* como de *software*, aunque será en el mundo del *software* donde, normalmente, realizará su aprendizaje y carrera profesional.

En este capítulo se hace una introducción a las computadoras y a los lenguajes de programación como herramienta de comunicación entre la máquina y el usuario. Una vez conseguida esta comunicación las computadoras son una de las herramientas más eficientes y potentes para resolver problemas de cualquier nivel de complejidad.

1.1. ORGANIZACIÓN DE UNA COMPUTADORA

Desde que en la década de los cuarenta se crearon las primeras computadoras (en España está más extendido el término «ordenador») con nombres tan míticos como: «computadora de Atanasoff-Berry, UNIVAC, ENIAC (tal vez la más conocida) o EDVAC, ha cambiado mucho la computación y su historia se ha ido labrando sobre la aparición de fechas o modelos concretos. Así podemos recordar el Apple construido por Steve Jobs y Stephen Wozniac, en 1976, junto con la creación de la casa Apple; otra fecha histórica es 1981, en la que IBM presentó a nivel mundial el/la PC (Personal Computer), Computador Personal. La aparición de IBM fue el punto de partida de la moderna informática. Su *software* original, creado por una joven compañía conocida por Microsoft, constituyó también el origen de la moderna industria de software. Hoy el término PC está ampliamente utilizado para referirse a cualquier computadora personal o de trabajo, aunque otros términos en inglés como *desktop* (computadora de escritorio) también se utilizan.

Desde un punto de vista simple, una computadora procesa datos y los convierte en información significativa. Aunque a primera vista puedan parecer sinónimos «datos» e «información» existe una gran diferencia entre ellos. Los datos (del plural *datum*, aunque la mayoría utiliza datos para singular y plural) constan de hechos en bruto y figuras. La *información*, por otra parte, son datos procesados. La información tiene datos significativos; los datos en su forma original (en bruto) no. La Figura 1.1 muestra el modelo fundamental de datos: los datos entran a la computadora (por una parte llamada *entrada*), la computadora procesa esos datos y la información significativa es el resultado (se presentan en una parte denominada *salida*).

Los datos, por sí mismos, no sirven para nada y no son útiles para las personas que los manipulan y necesitan tomar decisiones con ellos. Por ejemplo, el presidente de una empresa no necesita un informe detallado de las nóminas de los 2.000 empleados de su empresa si lo que desea es estudiar las tendencias o proyecciones en el tiempo de las citadas nóminas o sueldos. En su lugar, el presidente necesitará más información significativa, tal como nóminas totales por departamento, incrementos de la nómina durante el año pasado, nómina media del empleado comparada con otras empresas de la misma línea de negocios, etc. Una computadora que tenga almacenado un programa adecuado puede producir rápidamente la información que el presidente necesita a partir de los datos en bruto de la nómina.

Figura 1.1. Los datos de entrada se convierten en información significativa.

El programa es la fuerza conductora de cualquier tarea que hace una computadora. Un *programa* es una lista de instrucciones detalladas que indican a la computadora lo que ha de hacer. La computadora no puede hacer nada sin un programa. Es el trabajo del programador de escribir programas lo que influye en la computadora y tomar datos en bruto y transformarlos en información significativa para el usuario final, como puede ser el caso del presidente de la empresa, el objetivo final.

Los programas modernos producen información en muchos formatos. Estos programas reproducen música, «charlan» con otras computadoras a través de las líneas telefónicas (mediante dispositivos denominados *mózdems*) o teléfonos celulares (*móviles*) y otros dispositivos externos. La salida se envía a la pantalla y a la impresora que constituyen los dispositivos más usuales para acceso a la computadora; es decir, los datos de entrada y los datos de salida pueden ser, realmente, cualquier cosa, texto, dibujos, sonidos...

Una computadora consta de dos partes bien diferenciadas, *hardware* y *software*. El **hardware** consta de las partes físicas, tangibles de la computadora. El **software** consta de programas, también llamados *aplicaciones*, que contienen instrucciones que la computadora «ejecuta» o «corre»[1].

Una computadora necesita tanto del *hardware* como del *software* para poder ser usada real y prácticamente. El *hardware* de la computadora sin el *software* que le acompaña es como si tuviéramos un libro con páginas en blanco. La portada, contraportada y las páginas interiores constituyen el *hardware* del libro, pero el libro no es útil sin ningún texto, el *software*.

1.2. HARDWARE

Cuando un usuario interactúa con una computadora, proporciona una *entrada*; en respuesta, la computadora procesa la entrada devolviendo una *salida* valiosa al usuario. La entrada puede ser en formato de órdenes o instrucciones dadas, texto, números o imágenes *escaneadas*. La salida puede ser el resultado: cálculos en una hoja de cálculo, una carta impresa en una impresora o un auto moviéndose por la pantalla en un juego de carreras de autos. Una computadora necesita disponer de un conjunto de funcionalidades y proporcionar la capacidad de:

1. Aceptar la entrada.
2. Visualizar o presentar la salida.
3. Almacenar la información en un formato consistente lógicamente (tradicionalmente binario).
4. Ejecutar operaciones aritméticas o lógicas bien sobre datos de entrada o bien sobre datos de salida.
5. Monitorizar, controlar y dirigir las operaciones globales y de secuencia del sistema.

La Figura 1.2 representa los componentes más importantes que soportan estas propiedades o capacidades.

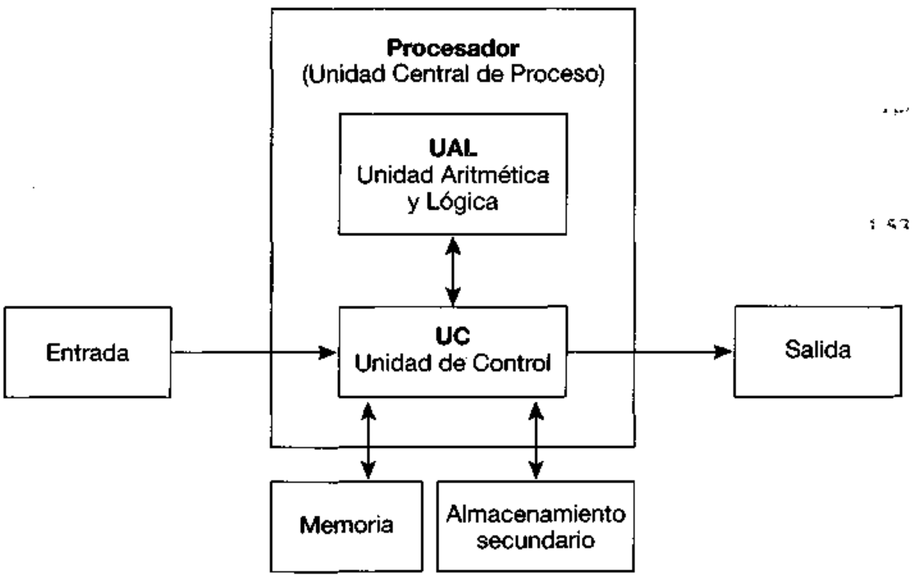


Figura 1.2. Unidades fundamentales de una computadora.

[1] En inglés, *execute* o *run*.

1.2.1. El procesador

El procesador es el dispositivo interior de la computadora que ejecuta las instrucciones del programa. Otros términos sinónimos utilizados para nombrar al procesador son: **UCP** (Unidad Central de Proceso, en inglés **CPU**, ***Central Processing Unit***) o chip. En el mercado existen muchos *chips* disponibles (Pentium, Athlon, etc.). El procesador sólo puede ejecutar instrucciones simples, tales como cálculos aritméticos sencillos o desplazamientos de números por diferentes posiciones. Sin embargo, la velocidad a la cual se realizan estos cálculos debe ser muy grande y esta característica le permite ejecutar instrucciones que realizan cálculos complejos.

La *Unidad Central de Proceso,* **UCP**, dirige y controla el proceso de información realizado por la computadora. La UCP procesa o manipula la información almacenada en memoria; puede recuperar información desde memoria (esta información son datos o instrucciones: programas). También puede almacenar los resultados de estos procesos en memoria para su uso posterior.

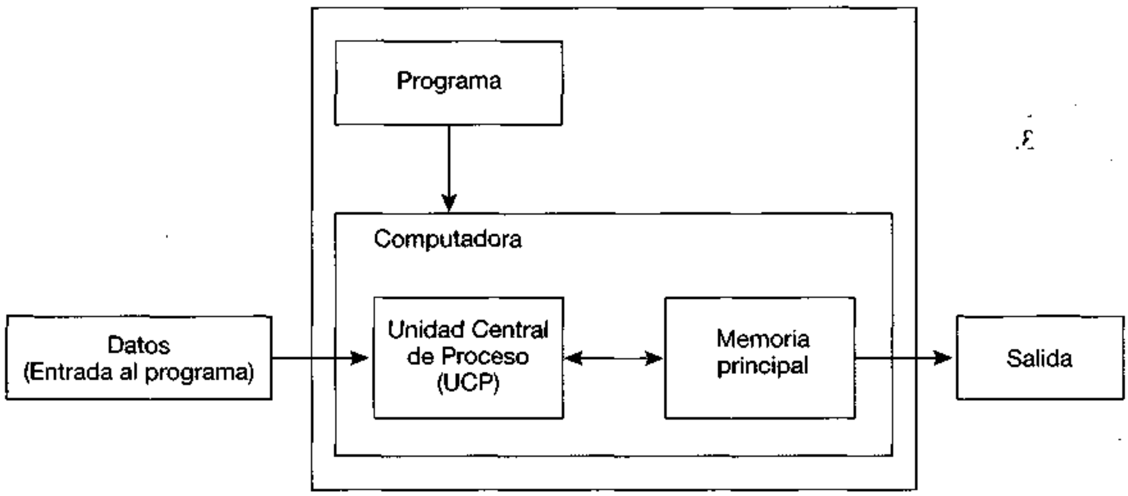


Figura 1.3. Ejecución de un programa en una computadora.

La **UCP** consta de dos componentes: *unidad de control* (**UC**) y *unidad aritmético-lógica* (**UAL**) (Fig. 1.3). La **unidad de control** (*Control Unit*, **CU**) coordina las actividades de la computadora y determina qué operaciones se deben realizar y en qué orden; asimismo controla y sincroniza todo el proceso de la computadora. La **unidad aritmético-lógica** (*Aritmethic-Logic Unit*, **ALU**) realiza operaciones aritméticas y lógicas, tales como suma, resta, multiplicación, división y comparaciones. Los datos en la memoria central se pueden *leer* (recuperar) o *escribir* (cambiar) por la UCP.

Otra parte muy importante de una computadora es la **memoria.** La *unidad memoria* almacena la información en un formato lógicamente consistente. Normalmente, tanto las instrucciones como los datos se almacenan en memoria, con frecuencia en áreas distintas y separadas. La memoria se divide en dos categorías: memoria principal y memoria auxiliar. La *memoria principal* (central) contiene el programa en ejecución y los resultados de los cálculos intermedios de la computadora. Se conoce normalmente como memoria RAM. La *memoria auxiliar* o almacenamiento secundario consta de dispositivos utilizados para almacenar los datos de modo permanente. Cuando se necesitan los datos se pueden recuperar de estos dispositivos. El programa se almacena en memoria externa de modo permanente pero cuando se ha de ejecutar debe transferirse a la memoria central. Este proceso se realiza mediante órdenes al sistema operativo que realiza las operaciones correspondientes.

1.2.2. El microprocesador

El **microprocesador** es un *chip* (circuito integrado) que controla y realiza las funciones y operaciones con los datos. En realidad, el microprocesador representa a la Unidad Central de Proceso o procesador, y popularmente cuando se habla de una computadora el término que se utiliza en las características técnicas para referirse a la Unidad Central de Proceso es procesador o microprocesador y con ese nombre y el modelo y marca correspondiente es como figura en los catálogos del fabricante.

La velocidad de un microprocesador se mide en megahercios (MHz) y manipulan palabras de 4 a 64 bits. Los microprocesadores históricos van desde el 8080 hasta el 80486/80586 pasando por el 8086, 8088, 80286 y 80386, todos ellos del fabricante Intel. Los fabricantes más populares son Intel, AMD, Cyrix y Transmeta, con modelos similares. Los microprocesadores de segunda generación de Intel son los Pentium, Pentium MMX, Pentium II, con velocidades de 233, 266, 300 y 450 MHz. Los microprocesadores de tercera generación son los **Pentium III**, con frecuencias de 450 y 500 MHz hasta 933 MHz y 1 GHz (modelo estándar en computadoras domésticas). Así mismo Athlon, del fabricante AMD, comercializa micros de hasta 1,1 GHz. Intel fabrica una serie económica y de menores prestaciones denominada *Celeron* que oferta procesadores de 500 MHz a 677 MHz.

Intel lanzó al mercado el año 2000 la siguiente generación, *Pentium 4*, con una nueva microarquitectura diseñada para ser más rápida y *escalar* mejor a los procesadores actuales; en la actualidad fabrica microprocesadores con frecuencias 1,3, 1,4, 1,5, 1,6 y 1,7 GHz y durante el mes de agosto de 2001 presentó su nuevo Pentium 4 a 2 GHz, que inmediatamente comenzó a ser incorporada a sus computadoras por marcas como Dell, Fujitsu o Compaq. AMD lanzó en julio de 2001 un modelo Duron a 1 GHz para PC de escritorio y el modelo Athlon a 1,4 GHz (según sus fabricantes superan al Intel de 1,7 GHz) y presentó a primeros del cuarto trimestre del 2001 su última versión de 1,5 GHz (para competir con el Pentium de 2 GHz) y anunció el Clawhammer de 64 bit a 2 GHz para este año. A finales de 2002, Intel presentó el Pentium 4 de 3 GHz y AMD presentó en octubre de 2002 el AMD XP 2800+ a.

Tabla 1.1. Relación de microprocesadores comerciales (enero 2003)

Intel		AMD	
Pentium MMX	233 MHz	Duron	900 MHz
Celeron	1,7 GHz	Duron	1,1 GHz
Celeron	1,8 GHz	Duron	1,3 GHz
Celeron	2,0 GHz		
Celeron	2,1 GHz		
Celeron	2,2 GHz		
Pentium 4	1,8 GHz	Athlon XP 1800+	1,53 GHz
Pentium 4	2,0 GHz	Athlon XP 2000+	1,66 GHz
Pentium 4	2,4 GHz	Athlon XP 2100+	1,73 GHz
Pentium 4	2,53 GHz	Athlon XP 2200+	1,8 GHz
Pentium 4	2,66 GHz	Athlon XP 2400+	2,0 GHz
Pentium 4	2,8 GHz	Athlon XP 2600+	2,08 GHz
Pentium 4	3,08 GHz	Athlon XP 2700+	2,16 GHz
		Athlon XP 2800+	2,25 GHz

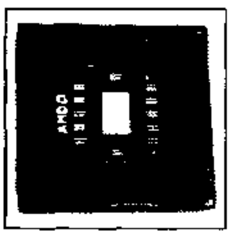


Figura 1.4. Micropocesadores Intel Pentium 4 a 3,06 GHz y Athlon XP 2800+.

Conviene recordar que el PC tal como se le conoce en la actualidad fue presentado por IBM el 12 de agosto de 1981 en Nueva York y otras ciudades norteamericanas: es decir, el PC tiene ya más de veinte años. Sin embargo, el PC no fue la primera computadora personal, ya que le precedieron otras máquinas con microprocesadores de 8 bits, muy populares en su tiempo, tales como *Apple II, PET CBM, Atari, TRS-80*, etc., y el mítico *ZX Spectrum.*

1.2.3. Dispositivos de Entrada/Salida (E/S)

Para que el usuario pueda introducir la entrada, la computadora tiene varios *dispositivos de entrada* como parte de su hardware: el teclado y el ratón son los más populares. La salida al usuario se le proporciona mediante *dispositivos de salida*, tales como impresora o pantalla. Posteriormente se verán otros dispositivos de entrada y/o salida.

Los dispositivos de *Entrada/Salida* (E/S) [*Input/Output* (**I/O**, en inglés)] permiten la comunicación entre la computadora y el usuario. *Los dispositivos de entrada*, como su nombre indica, sirven para introducir datos (información) en la computadora para su proceso. Los datos se *leen* de los dispositivos de entrada y se almacenan en la memoria central o interna. Los dispositivos de entrada convierten la información de entrada en señales eléctricas que se almacenan en la memoria central. Dispositivos de entrada típicos son los **teclados**; otros son **lectores de tarjetas** —ya en desuso—, **lápices ópticos, palancas de mando** (*joystick*), **lectores de códigos de barras, escáneres, micrófonos**, etc. Hoy día tal vez el dispositivo de entrada más popular es el **ratón** (*mouse*) que mueve un puntero electrónico sobre la pantalla que facilita la *interacción usuario-máquina*[2].

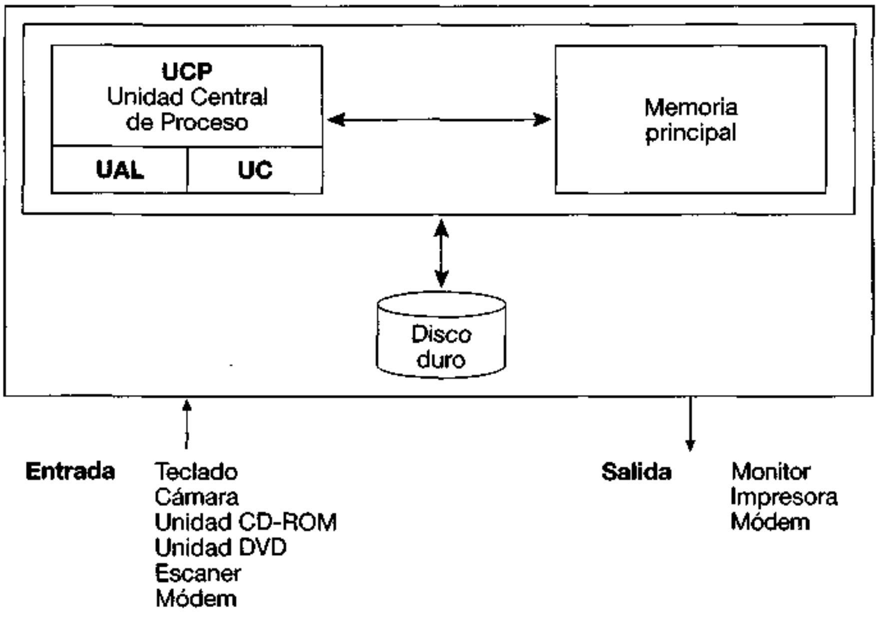


Figura 1.5. Diagrama de bloques de una computadora.

[2] Todas las acciones a realizar por el usuario se realizarán con el ratón, con la excepción de las que se requieren de la escritura de datos por teclado. El nombre de ratón parece proviene de la similitud del cable de conexión con la cola de un ratón. Hoy día, sin embargo, este razonamiento carece de sentido, ya que existen ratones inalámbricos que no usan cable y se comunican entre sí a través de rayos infrarrojos por ondas radio.

Los *dispositivos de salida* permiten representar los resultados (salida) del proceso de los datos. El dispositivo de salida típico es la **pantalla (CRT)**[3] o **monitor.** Otros dispositivos de salida son **impresoras** (imprimen resultados en papel), **trazadores gráficos** (*plotters*), **reconocedores de voz, altavoces**, etc.

El teclado y la pantalla constituyen, en muchas ocasiones, un único dispositivo denominado **terminal**. Un teclado de terminal es similar al teclado de una máquina de escribir moderna con la diferencia de algunas teclas extras que tiene el terminal para funciones especiales. Si está utilizando una computadora personal, el teclado y el monitor son dispositivos independientes conectados a la computadora por cables. En ocasiones a la impresora se la conoce como **dispositivo de copia dura** (*«hard copy»*), debido a que la escritura en la impresora es una copia permanente (dura) de la salida, y a la pantalla se le denomina en contraste: **dispositivo de copia blanda** (*«soft copy»*), ya que se pierde la pantalla actual cuando se visualiza la siguiente.

Los dispositivos de entrada/salida y los dispositivos de almacenamiento secundario o auxiliar (memoria externa) se conocen también con el nombre de *dispositivos periféricos* o simplemente **periféricos**, ya que, normalmente, son externos a la computadora. Estos dispositivos son unidad de discos (*disquetes, CD-ROM, DVDs*, cintas, etc.), videocámaras, etc.

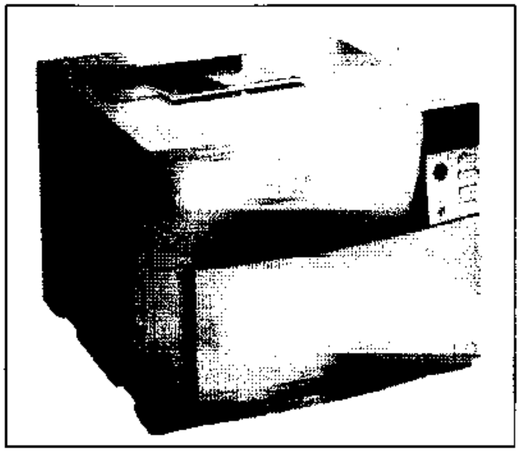


Figura 1.6. Dispositivo de salida (impresora).

1.2.4. La memoria central (interna)

Con el propósito de almacenar datos, una computadora consta de una gran colección de pequeños circuitos electrónicos capaces de almacenar **un bit**. Estos circuitos son como pequeños interruptores de luz que puedan estar en dos estados: uno para «encendido» (*on*) y otro para «apagado» (*off*).

Nota

Un bit es un circuito o un dígito que puede tener exactamente dos valores: cero o uno.

La memoria central o simplemente **memoria** (*interna* o *principal*) se utiliza para almacenar información). Se divide a su vez en memoria RAM y memoria ROM. La memoria **RAM** (**R***andom*, **Ac**-

[3] *Cathode Ray Tube*: Tubo de rayos catódicos.

cess Memory) es normalmente volátil. lo que significa que todo cuanto se almacena o guarda en ella se «pierde» cuando se apaga la computadora. En general, la información almacenada en memoria puede ser de dos tipos: las *instrucciones* de un programa y los *datos* con los que operan las instrucciones. Por ejemplo, para que un programa se pueda *ejecutar* (correr, rodar, funcionar..., en inglés *run*), debe ser situado en la memoria central, en una operación denominada *carga* (*load*) del programa. Después, cuando se ejecuta (se realiza, funciona) el programa, *cualquier dato a procesar por el programa se debe llevar a la memoria* mediante las instrucciones del programa. En la memoria central hay también datos diversos y espacio de almacenamiento temporal que necesita el programa cuando se ejecuta con él a fin de poder funcionar. Los programas y datos se almacenan en RAM mientras se está utilizando la computadora. La memoria **RAM** puede ser *estática* o *dinámica* según sea el proceso de fabricación. Las memorias actuales se clasifican en **DRAM**, **SDRAM**, **EDRAM**, etc.

Con el objetivo de que el procesador pueda obtener los datos de la memoria central más rápidamente, la mayoría de los procesadores actuales (muy rápidos) utilizan con frecuencia una *memoria* denominada *caché*, que sirve para almacenamiento intermedio de datos entre el procesador y la memoria principal. La memoria *caché* —en la actualidad— se incorpora casi siempre al procesador.

La memoria central de una computadora es una zona de almacenamiento organizada en centenares o millares de unidades de almacenamiento individual o celdas. La memoria central consta de un conjunto de *celdas de memoria* (estas celdas o posiciones de memoria se denominan también *palabras*, aunque no «guardan» analogía con las palabras del lenguaje). El número de celdas de memoria de la memoria central, dependiendo del tipo y modelo de computadora; hoy día el número suele ser millones (32, 64, 128, etc.). Cada celda de memoria consta de un cierto número de bits (normalmente 8, un *byte*).

La unidad elemental de memoria se llama *byte* (octeto). Un *byte* tiene la capacidad de almacenar un carácter de información, y está formado por un conjunto de unidades más pequeñas de almacenamiento denominadas *bits*, que son dígitos binarios (0 o 1).

El tamaño de la RAM de una computadora normalmente se especifica en bytes disponibles al usuario. La memoria en un PC varía en el rango de 64/128 millones de bytes a 256/512 millones de bytes.

La memoria **ROM** (Read Only Memory), *memoria de solo lectura*, contiene instrucciones fundamentales que no se pueden modificar o perder de modo accidental por el usuario. Estas memorias incluyen aquellas instrucciones que son necesarias para cargar inicialmente al software cuando se enciende («*se prende*») la máquina y cualesquiera otras instrucciones que el fabricante requiere para estar accesible desde un modo personal cuando el computador está funcionando. La ROM es no volátil, su contenido no se pierde cuando se apaga la energía. Al ser esta memoria de solo lectura, los programas almacenados en los chips ROM no se pueden modificar y suelen utilizarse para almacenar los programas del sistema que sirven para arrancar la computadora.

1.2.5. Direcciones de memoria

Existen dos conceptos importantes asociados a cada celda o posición de memoria: su *dirección* y su *contenido*. Cada celda o byte tiene asociada una única *dirección* que indica su posición relativa en memoria y mediante la cual se puede acceder a la posición para almacenar o recuperar información. La información almacenada en una posición de memoria es su *contenido*. La Figura 1.7 muestra una memoria de computadora que consta de 1.000 posiciones en memoria con direcciones de 0 a 999. El contenido de estas direcciones o posiciones de memoria se llaman *palabras*, de modo que existen palabras de 8, 16, 32 y 64 bits. Por consiguiente, si trabaja con una máquina de 32 bits, significa que en cada posición de memoria de su computadora puede alojar 32 bits, es decir, 32 dígitos, bien ceros o unos.

Siempre que una nueva información se almacena en una posición. se destruye (desaparece) cualquier información que en ella hubiera y no se puede recuperar. La dirección es permanente y única, el contenido puede cambiar mientras se ejecuta un programa.

La memoria central de una computadora puede tener desde unos centenares de millares de bytes hasta millones de bytes. Como el byte es una unidad elemental de almacenamiento, se utilizan múlti-

plos para definir el tamaño de la memoria central: *Kilo*-byte (**KB**) igual a 1.024 bytes (2^{10}) —prácticamente se consideran 1.000— y *Megabyte* (**MB**) igual a 1.024 × 1.024 bytes (2^{20}) —prácticamente se considera un 1.000.000— y *Gigabyte* (**GB**) igual a 1.024 MB.

Tabla 1.1. Unidades de medida de almacenamiento.

Byte	**Byte (B)**	*equivale a*	8 bits
Kilobyte	**Kbyte (KB)**	*equivale a*	1.024 bytes (2^{10})
Megabyte	**Mbyte (MB)**	*equivale a*	1.024 Kbytes (2^{20})
Gigabyte	**Gbyte (GB)**	*equivale a*	1.024 Mbytes (2^{30})
Terabyte	**Tbyte (TB)**	*equivale a*	1.024 Gbytes (2^{40})
Petabyte	**Pbyte (PB)**	*equivale a*	1.024 Tbytes (2^{50})
Exabyte	**Ebyte (EB)**	*equivale a*	1.024 Pbytes (2^{60})
Zettabyte	**Zbyte (ZB)**	*equivale a*	1.024 Ebytes (2^{70})
Yotta	**Ybyte (YB)**	*equivale a*	1.024 Ybytes (2^{80})

`1 Tb = 1.024Gb   Gb = 1.024Mb   Mb = 1.048.576b   Kb = 1.073.741.824 b`

En la actualidad las computadoras personales domésticas tipo PC suelen tener memorias centrales cuyo rango varía entre 128 y 256 MB a 512 MB.

Bytes y direcciones

La memoria principal se divide en posiciones numeradas denominadas *bytes*. El número asociado con un byte es su *dirección* en alusión a la técnica de identificar las casas de una ciudad por su dirección. Un grupo de bytes consecutivos se utiliza como la posición de un elemento de datos, tal como un número o letra. La dirección del primer byte del grupo se utiliza como la dirección de esta posición de memoria más alta.

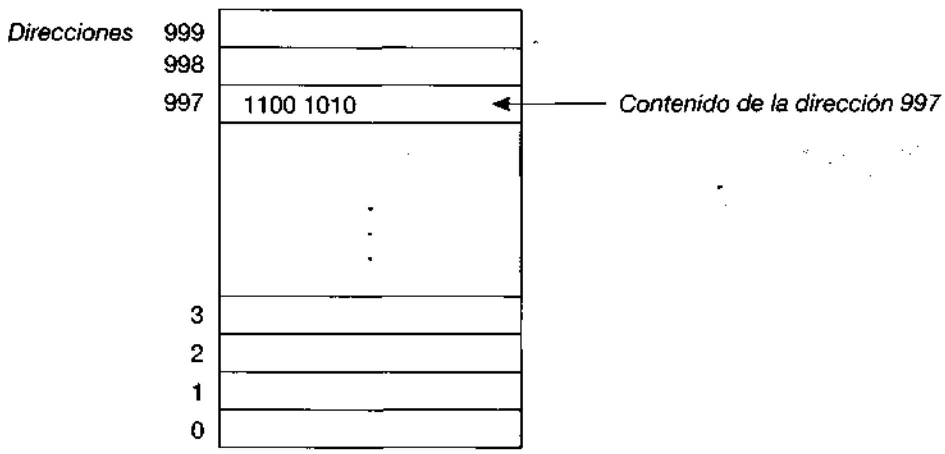


Figura 1.7. Memoria central de una computadora.

La memoria principal es la encargada de almacenar los programas y datos que se están ejecutando y su principal característica es que el acceso a los datos o instrucciones desde esta memoria es muy rápido.

En la memoria principal se almacenan:

- Los datos enviados para procesarse desde los dispositivos de entrada.
- Los programas que realizarán los procesos.
- Los resultados obtenidos preparados para enviarse a un dispositivo de salida.

1.2.6. Memoria auxiliar (externa)

Cuando un programa se ejecuta, se debe situar primero en memoria central de igual modo que los datos. Sin embargo, la información almacenada en la memoria se pierde (borra) cuando se *apaga* (desconecta de la red eléctrica) la computadora y, por otra parte, la memoria central es limitada en capacidad. Por esta razón, para poder disponer de almacenamiento permanente, tanto para programas como para datos, se necesitan *dispositivos de almacenamiento secundario, auxiliar* o *masivo* (*«mass storage»*, o *«secondary storage»*).

Los **dispositivos de almacenamiento** o **memorias auxiliares** (*externas* o *secundarias*) más comúnmente utilizados son: *cintas magnéticas, discos magnéticos, discos compactos* (**CD-ROM** *Compact* **D***isk* **R***ead* **O***nly* **M***emory*), y *videodiscos digitales* o *discos versátiles digitales* (**DVD**, **D***igital* **V***ersatile* **D***isc*). Las cintas son utilizadas principalmente por sistemas de computadoras grandes similares a las utilizadas en los equipos de audio. Los discos y disquetes magnéticos se utilizan por todas las computadoras, especialmente las de tamaño medio y pequeño —las computadoras personales—. Los discos pueden ser **duros**, de gran capacidad de almacenamiento (su capacidad actual suele variar de unos 20-30 GB a unos 100 GB), **disquetes** o *discos flexibles* («floppy disk») (360 KB a 1,44 MB). El tamaño físico de los disquetes, y por el que son conocidos, es de 5¼ pulgadas (5,25") (ya en desuso total), 3½ (3,5") (en pocos años caerá también en desuso). Las dos caras de los discos se utilizan para almacenar información. La capacidad de almacenamiento varía en función de la intensidad de su capa ferromagnética y pueden ser de doble densidad (***DD***) o de alta densidad (***HD***). El disquete usual suele ser de 3,5" y de 1,44 MB de capacidad.

Los discos duros suelen estar fijos en los sistemas y normalmente no se pueden extraer de sus unidades, aunque cada vez es más frecuente encontrar discos duros extraíbles (*removibles*), sobre todo con la finalidad de preparar discos para copias de seguridad.

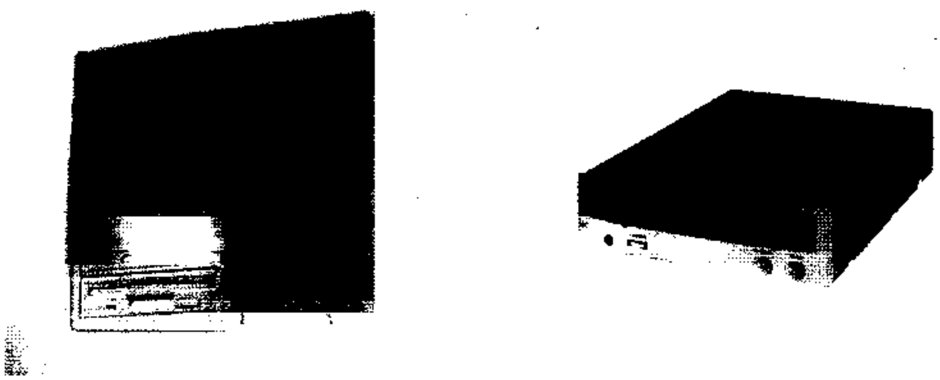


Figura 1.8. Memorias auxiliares: *a*) disco duro de 3½"; *b*) unidad y lector ZIP.

Existen otros tipos de dispositivos de almacenamiento que se almacenan en unas unidades especiales denominadas *zip* que tienen gran capacidad de almacenamiento comparada con los disquetes tra-

dicionales de 1,44 MB. Estos disquetes son capaces de almacenar 100 MB, aunque ya se comercializan unidades de discos *zip* de 250 MB y han salido en el último trimestre del 2002 las unidades de 750 MB.

La información almacenada en la memoria central es *volátil* (desaparece cuando se apaga la computadora) y la información almacenada en la memoria auxiliar es *permanente*. Esta información se organiza en unidades independientes llamadas **archivos** (**ficheros**, *file* en inglés). Los resultados de los programas se pueden guardar como *archivos de datos* y los programas que se escriben se guardan como *archivos de programas*, ambos en la memoria auxiliar. Cualquier tipo de archivo se puede transferir fácilmente desde la memoria auxiliar hasta la memoria central para su proceso posterior.

La memoria auxiliar permanente almacena gran cantidad de archivos en los diferentes dispositivos mencionados anteriormente (discos flexibles, disquetes, etc.). Cada archivo normalmente tiene un nombre arbitrario (por ejemplo, `MisDocumentos`, `Ciudad`) y una extensión que indica el tipo de archivo que se está tratando (tales como `.doc`, que indica un archivo de un procesador de textos, Word). Así el archivo Word citado anteriormente sería `MisDocumentos.doc`.

Los archivos pueden contener cualquier tipo de datos. Ejemplos, pueden ser una imagen, un archivo de sonido, un programa de computadora, una letra o simplemente una lista de números. Se utilizan diversas anotaciones para codificar e interpretar los contenidos de un archivo. Así hay archivos multimedia `MPEG` (extensión `.mpg`), archivos `.JPEG` para almacenar imágenes (extensión `.jpg`).

Un tipo específico de archivo se encuentra cuando se escribe el programa en un lenguaje de programación. Por ejemplo, en el caso de C la extensión es `.c` y si estamos en C# la extensión será `cs` (*c sharp*). Así, un archivo escrito en código C podría tener el nombre de `miPrograma.c`. Los archivos se disponen normalmente en grupos de archivos denominados carpetas (*folders*) o directorios.

En el campo de las computadoras es frecuente utilizar la palabra memoria y almacenamiento o memoria externa, indistintamente. En este libro —y recomendamos su uso— se utilizará el término memoria sólo para referirse a la memoria central.

Comparación de la memoria central y la memoria auxiliar

La memoria central o principal es mucho más rápida y cara que la memoria auxiliar. Se deben transferir los datos desde la memoria auxiliar hasta la memoria central, antes de que puedan ser procesados. Los datos en memoria central son: *volátiles* y desaparecen cuando se apaga la computadora. Los datos en memoria auxiliar son *permanentes* y no desaparecen cuando se apaga la computadora.

Las computadoras modernas necesitan comunicarse con otras computadoras. Si la computadora se conecta con una *tarjeta de red* se puede conectar a una red de datos locales (*red de área local*). De este modo se puede acceder y compartir a cada una de las memorias de disco y otros dispositivos de entrada y salida. Si la computadora tiene un *modem*, se puede comunicar con computadoras distantes. Se pueden conectar a una red de datos o *enviar correo electrónico* a través de las redes corporativas *Intranet/Extranet* o la propia red *Internet*. También es posible enviar y recibir mensajes de fax.

1.3. DISPOSITIVOS DE ALMACENAMIENTO DE INFORMACIÓN

Las unidades, periféricos o dispositivos de almacenamiento son dispositivos periféricos que actúan como medio de soporte para almacenar los datos —temporal o permanentemente— que ha de manipular la CPU durante el proceso en curso y que no puede contener la memoria principal. Las tecnologías han evolucionado muy rápidamente y se han conseguido dispositivos de almacenamiento masivos de datos. Existe una variedad grande de dispositivos de almacenamiento que se clasifican en dos grandes grupos: *discos* y *cintas magnéticas*.

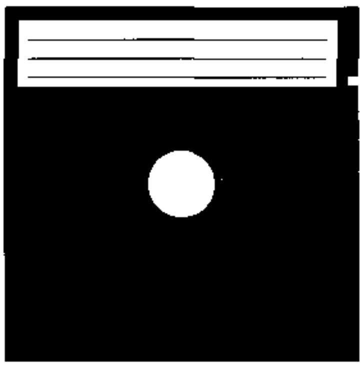
1.3.1. Discos

Los discos son dispositivos formados por componentes electromagnéticos o por componentes ópticos que permiten un acceso rápido a bloques físicos de datos. La información se registra en la superficie del disco y se accede a ella por medio de cabezas de *lectura/escritura* que se mueven sobre la superficie. Su principal característica es el acceso directo a la información y se les conoce también como *unidades de almacenamiento masivo* o *memoria masiva*. Los discos magnéticos —ya citados anteriormente— se clasifican en **disquetes** y **discos duros**, y los discos ópticos en **CD-ROM** y **DVD**.

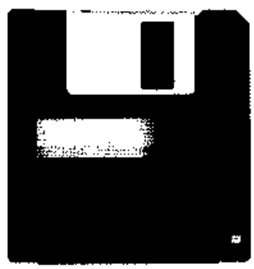
Disquetes

Los **disquetes** o discos flexibles (*floppy disks*) son discos realizados en material flexible que son transportables de una computadora a otra. Las principales ventajas de estos discos, además de la *transportabilidad* o *portabilidad*, son su precio y la compatibilidad (universalidad) con cualquier computadora. Se suelen utilizar para almacenar programas y archivos que se utilizan con poca frecuencia para transferir archivos de un PC a otro y como medio de seguridad (respaldo) de los datos contenidos en los discos duros. Por el contrario, carecen de gran capacidad, así como de velocidad e incluso fiabilidad, comparados con los discos duros o los discos ópticos.

Los primeros disquetes, antes del advenimiento del PC, eran de 8 pulgadas ("); posteriormente aparecieron del tamaño de 5 ¼ de 360 KB de capacidad que llegaron a alcanzar 1,2 MB (ya prácticamente en desuso). La última innovación de 3,5" y capacidad de 1,44 Megabytes es hoy día el formato más utilizado, aunque ya tienen una gran competencia en los discos ópticos del tipo CD-ROM.



a)



b)

Figura 1.9. Disquetes: *a)* 5,25"; *b)* 3,5".

Discos duros

A principios de la década de los ochenta surgieron los primeros discos duros o fijos y con una capacidad de 10 MB, cantidad por aquella época más que suficiente para alojar el sistema operativo y las aplicaciones más utilizadas[4]. Los **discos duros** están construidos con material rígido sobre el cual se deposita una película de material magnetizable que permite la grabación de datos. Estos discos permiten mayores densidades de almacenamiento y tiempos de accesos a los datos almacenados en ellos muy inferiores a los disquetes. Los discos duros están constituidos en su interior por unos platos en los que se almacena la información. Los últimos modelos de discos duros tienen tamaños pequeños,

---

[4] Estas cifras son hoy día normales para simples archivos musicales tipo MP3 o unos cuantos archivos tipo procesador de textos Word u hoja de cálculo Excel.

incluso del orden de tarjetas de crédito o cajas de cerillas, pero, sin embargo, tienen grandes capacidades de almacenamiento (20, 30, 40, ... gigas).

> Los discos duros ofrecen una buena relación capacidad de almacenamiento (coste y tiempos de acceso muy rápidos.

Existen dos tipos de discos duros: *Fijos* (se fabrican en el interior de una carcasa sellada de la que no se pueden extraer). *Removibles* (se fabrican también en una carcasa contenedora, también sellada, pero se pueden introducir en la unidad de proceso de la computadora, bien en ranuras adecuadas o bien mediante cables especiales); estos discos se suelen utilizar para el traslado de grandes cantidades de información de una computadora a otra.

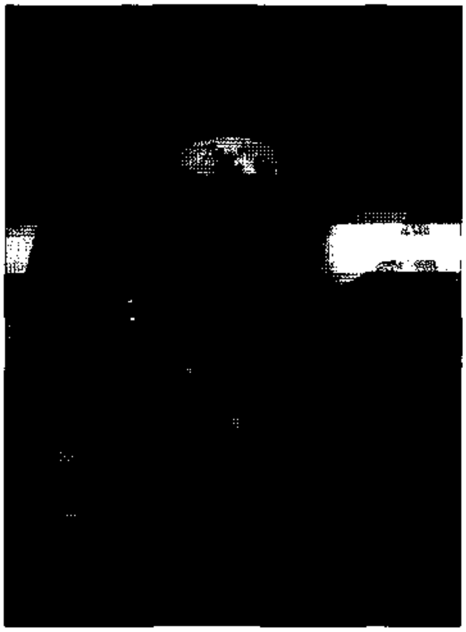


Figura 1.10. Discos duros.

Discos duros virtuales

Es un nuevo dispositivo de almacenamiento de información que no reside en la computadora del usuario sino en un espacio virtual residente en un sitio Web de Internet (de su propia empresa, o de cualquiera otra que ofrezca el servicio). Es una buena opción para el usuario (estudiantes, particulares, profesionales, empresas...) de tipo medio y empresas que utilizan grandes volúmenes de información y que necesitan más espacio y no lo tienen disponible en sus equipos. Este almacenamiento o alojamiento puede ser gratuito o de pago, pero en cualquier forma no deja de ser una interesante oferta para el programador, que encuentra un lugar donde situar aplicaciones, archivos, etc., que no puede almacenar en su computadora.

El inconveniente de esta solución es el riesgo que se ha de asumir de depositar información en lugares no controlados por uno mismo. Esta situación plantea la necesidad de un estudio de la privacidad y seguridad que van a tener los datos que deposite en estos discos virtuales. La Tabla 1.2 muestra

algunas direcciones de almacenamiento virtual en Internet que en el momento de la escritura de esta obra eran gratuitos, al menos durante un período de prueba.

Tabla 1.2. Direcciones de sitios Web para almacenamiento virtual de datos (*visitadas* noviembre 2002)

Nombre de la empresa	Dirección de Internet (URL)	Observaciones
Xdrive	`www.xdrive.com`	
FreDrive	`www.xdrive.com`	Fue comprada por Xdrive
FreeMailGuide	`www.freemailguide.com`	Correo electrónico gratuito
Go Daddy	`www.sharenation.com`	
Yahoo! Briefcase	`briefcase.yahoo.com`	Sitio gratuito de Yahoo

1.3.2. Discos ópticos: CD-ROM y DVD

Los **discos ópticos** difieren de los tradicionales discos duros o discos magnéticos en que los primeros utilizan un haz de láser para grabar la información. Son dispositivos de almacenamiento que emplean la misma tecnología que los dispositivos compactos de audio para almacenar información digital. Por esta razón suelen tener las mismas características que los discos de música: muy resistentes al paso del tiempo y con gran capacidad de almacenamiento. Estos discos se suelen utilizar para almacenar información histórica (no va a sufrir modificaciones frecuentes), archivos gráficos complejos, imágenes digitales, etc. Al igual que los disquetes, son transportables y compatibles entre computadoras.

El CD-ROM (El Cederrón)[5]

El **CD-ROM** (*Compact Disk-Read Only Memory, Disco compacto - Memoria de solo lectura*). Estos discos son el medio ideal para almacenar información de forma masiva que no necesita ser actualizada con frecuencia (dibujos, fotografías, enciclopedias...). La llegada de estos discos al mercado hizo posible el desarrollo de la *multimedia*, es decir, la capacidad de integrar medios de todo tipo (texto, imágenes, sonido e imágenes). Permiten almacenar 650 o 700 Megabytes de información. En la actualidad son muy económicos, alrededor de 0,40 a 0,50 €. Estos discos son de sólo lectura, por lo que sólo se pueden grabar una vez y son conocidos como **CD-R**, siendo cada día más populares, y lo normal será que estos modelos sustituyan en breve plazo a los disquetes rígidos de 3,5".

CD-RW (Recordable y ReWritable)

Existen discos que permiten grabación de datos, además de lectura, y se conocen como discos **CD-RW** (CD-*Recordable y ReWritable*). Desde hace unos años es posible encontrar en el mercado estos discos ópticos CD en los que se puede leer y escribir información por parte del usuario cuantas veces se desee. Es el modelo *regrabable*, por excelencia. Este modelo se suele utilizar para realizar copias de seguridad del disco duro o de la información más sensible, al poder actualizarse continuamente. Aunque nació para emplearse en servidores, estaciones de trabajo, etc., hoy día es un disco que suele utilizarse en computadoras personales de grandes prestaciones. Las unidades lectoras y grabadoras de discos[6] de este tipo tiene ya precios asequibles y son muchos los usuarios, incluso domésticos, que incorporan estas unidades a sus equipos informáticos.

[5] La última edición (22.ª, 2001) del Diccionario de la Lengua Española (Real Academia Española) ha incorporado el término **cederrón**.

[6] En Latinoamérica se conoce también a estas unidades como unidades «**quemadoras**» de disco, traducción fiel del término anglosajón.

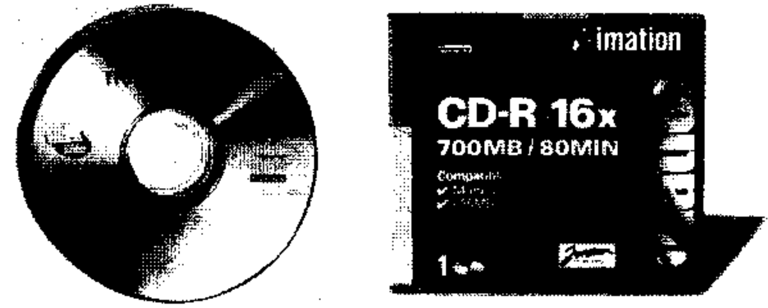


Figura 1.11. Discos CD-R y CD-RW.

DVD (Digital Vesatil Disk): Videodisco digital

Este disco óptico nació en 1995 gracias a un acuerdo entre los grandes fabricantes de electrónica de consumo, estudios de cine y de música (Toshiba, Philips, Hitachi, JVC...). Son dispositivos de alta capacidad de almacenamiento, interactivos y con total compatibilidad con los medios existentes. Tiene además una gran ventaja: su formato sirve tanto para las computadoras como para los dispositivos de electrónica de consumo. Es capaz de almacenar hasta 26 CD con una calidad muy alta y con una capacidad que varía, desde los 4,7 GB del tipo de una cara y una capa hasta los 17 GB de la de dos caras y dos capas, o lo que es igual, el equivalente a la capacidad de 7 a 26 CDs convencionales. Estas cifras significan que se pueden almacenar en uno de estos discos una película completa en diferentes idiomas e incluso subtítulos

En la actualidad se pueden encontrar tres formatos de DVD grabables: **DVD-R** (se puede grabar una sola vez); **DVD-RAM** (*re-escribible* pero con un funcionamiento similar al disco duro); **DVD-RW** (lectura y escritura, regrabable). Al igual que en el caso de los discos compactos, requieren de unas unidades especiales de lectura y reproducción, así como grabadoras/reproductoras. Estas últimas se encuentran ya en el mercado, pero sus precios todavía no son asequibles (Philips, por ejemplo, es ya un fabricante que comercializa unidades lectoras/grabadoras de discos DVD).

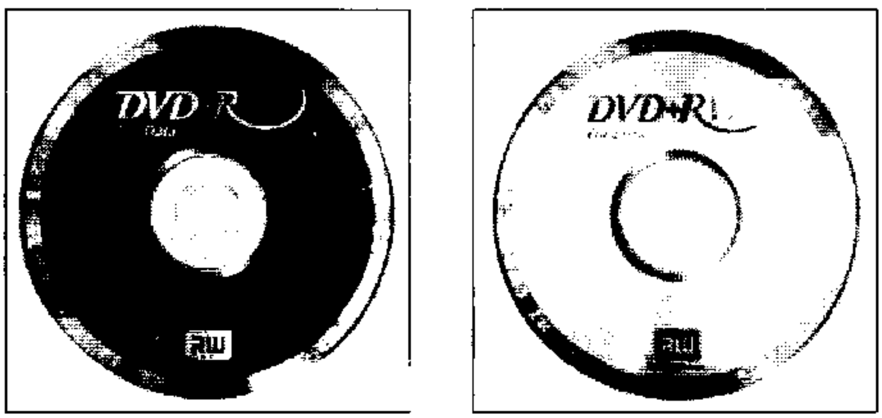


Figura 1.12. Unidad de discos DVD.

Cintas

Las **cintas magnéticas** son los primeros dispositivos de almacenamiento de datos que se utilizaron y, por ello, hasta hace poco tiempo —y aún hoy—, han sido los más empleados para almacenar copias de se-

guridad. Poseen una gran capacidad de almacenamiento pero tienen un gran inconveniente, son *dispositivos de almacenamiento de acceso secuencial,* es decir, no soportan el acceso directo (aleatorio) a los datos debido a que su unidad de lectura debe explorar la cinta hasta encontrar la información específica. Por esta razón, la rapidez de acceso a los datos en las cintas es menor que en los discos. A mayor capacidad de almacenamiento, mayor longitud de cinta, y, en consecuencia, mayor tiempo de acceso.

Por las causas anteriores, la información que no requiere procesarse inmediatamente, es decir, que no necesita estar en línea, puede almacenarse en cintas. Éstas pueden utilizarse como medio de soporte para realizar copias de seguridad de discos duros y como soporte para el almacenamiento de grandes bases de datos. En general, las cintas son medios removibles, fiables y económicos, y proporcionan capacidades altas de almacenamiento.

Para compensar los inconvenientes citados, se han desarrollado cartuchos de cintas (*streamers*) que realizan las mismas funciones que las cintas, pero en un espacio menor y con mejoras en prestaciones, facilidad de manejo y seguridad, aunque siguen siendo medios de acceso secuencial. Son muy similares a los casetes de audio, aunque con diferentes tamaños.

Cintas de audio digital: DAT

Las cintas de audio digital (**DAT**, **D***igital* **A***udio* **T***ape*) son unidades de almacenamiento con capacidad para grabar varios **gigabytes** de información en un único cartucho. Son dispositivos de pequeñas dimensiones, económicos y rápidos; sin embargo, sus unidades lectoras son caras. Estas cintas se utilizan en las mismas aplicaciones que las cintas de cartucho (*streamer*) como medio de copia de seguridad o respaldo (*backup*) pero con características que permiten disponer de mayores capacidades de almacenamiento tanto para estaciones de trabajo como para servidores. Aunque las cintas DAT ofrecen mayores capacidades de almacenamiento, su precio es más elevado. Sin embargo, para los usuarios con necesidades de almacenamiento de copias de seguridad grandes son una solución adecuada.

Disco Flash

Un **disco flash** (*flash disk*) es un pequeño almacén de memoria móvil de un tamaño algo mayor que un mechero o una microlinterna y que, por consiguiente, se puede transportar como un llavero. Este disco se puede utilizar en cualquier computadora que disponga de una conexión (puerto) USB. No necesita suministro de corriente adicional. Se comercializa por numerosos fabricantes (entre ellos Fujitsu) y pueden almacenar desde 16 MB hasta 256 MB y con unos precios módicos que pueden oscilar de unos 20 euros a unos 100 euros según capacidad.

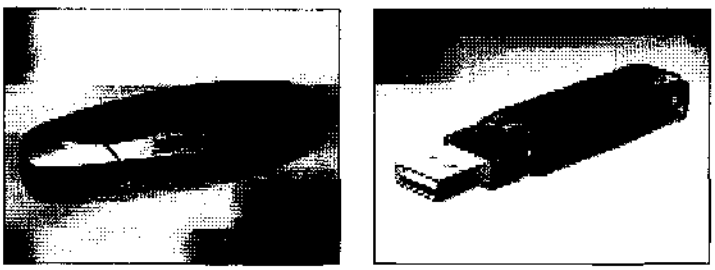


Figura 1.13. Disco flash.

1.4. LA COMPUTADORA PERSONAL IDEAL PARA PROGRAMACIÓN

Hoy día el estudiante de informática o de computación, y mucho más el profesional, dispone de un amplio abanico de computadoras a precios asequibles y con prestaciones altas. En el último trimestre del año 2002, un PC de escritorio típico para aprender a programar y posteriormente utilizar de modo

profesional era posible encontrarlo con precios comprendidos en el rango entre 1.000 y 1.500 €[7], dependiendo de prestaciones y fabricante. La Tabla 1.3 resume nuestra propuesta y recomendación de características medias en un PC, sobre la base de finales del 2002.

Tabla 1.3. Características de un PC ideal.

Procesador	Microprocesador de la marca Intel, AMD o Transmeta (en este caso sólo para computadores portátiles, «laptop»). Un mínimo de frecuencia de 1,5 GHz, recomendable Pentium 4, mínimo 2,4/2,53 GHz o Athlon de 1,53 GHz En ambientes profesionales ya es una realidad el Pentium 4 de 3 GHz.
Memoria	128 MB mínima y recomendable para aplicaciones profesionales 256 a 512 MB.
Caché	Memoria especial que usa el procesador para acelerar sus operaciones. 256 KB a 512 KB.
Disco duro	30 a 60 GB. Para fines profesionales, un mínimo de 80 GB.
Internet	Preparado para Internet (preferentemente con módem instalado de 56 KB). Conexión por cable o línea telefónica de cobre con tecnología ADSL. Tarjeta de Red. Posibilidad para conexión a red inalámbrica y tecnología *Bluetooth*.
Vídeo (RAM instalada)	Memoria de vídeo, con un mínimo de 64 MB.
Monitor	15", 17", 19", 21". El tamaño más popular es de 15", aunque la tendencia actual es a monitores de 17". Las pantallas planas cada vez se están haciendo más populares, a medida que se abarata su precio; 15" es el tamaño ideal, si bien 17" mejora considerablemente la visión.
Almacenamiento	CD-ROM, DVD-ROM. Preferentemente debería incorporar una unidad grabadora (quemadora) de discos CD; es decir, una unidad de discos CD-RW (velocidad recomendable, 32 × 12 × 40).
Tarjeta gráfica	Recomendable Nvidia GeForce 2 4Mx400 con 64 MB de SDRAM o similar.
Puertos	Serie, paralelo y USB.
Marcas	HP, Compaq, Dell, IBM, Fujitsu Siemens, El System, Jump, ADL, Gateway, etc.
Sistema Operativo	Windows 98 / Me / NT/ 2000; Windows XP; Linux; Solaris; Macintosh (preferentemente Windows XP Home/Professional).

1.5. EL SOFTWARE (LOS PROGRAMAS)

Las operaciones que debe realizar el *hardware* son especificadas por una lista de instrucciones, llamadas programas o *software*. El software se divide en dos grandes grupos: *software del sistema* y *software de aplicaciones*.

El **software del sistema** es el conjunto de programas indispensables para que la máquina funcione; se denominan también *programas del sistema*. Estos programas son, básicamente, el *sistema operativo*, los *editores de texto*, los *compiladores/intérpretes* (lenguajes de programación) y los *programas de utilidad*.

Uno de los programas más importante es el **sistema operativo**, que sirve, esencialmente, para facilitar la escritura y uso de sus propios programas. El sistema operativo dirige las operaciones globales de la computadora, instruye a la computadora para ejecutar otros programas y controla el almacenamiento y recuperación de archivos (programas y datos) de cintas y discos. Gracias al sistema operativo es posible que el programador pueda introducir y grabar nuevos programas, así como ins-

[7] La conversión «euro-dólar» en el último trimestre de 2002 se hacía aproximadamente a la paridad; en febrero de 2003, 1 € ≅ 1,1 $.

truir a la computadora para que los ejecute. Los sistemas operativos pueden ser: *monousuarios* (un solo usuario) y *multiusuarios*, o tiempo compartido (diferentes usuarios), atendiendo al número de usuarios, y *monotarea* (una sola tarea) o *multitarea* (múltiples tareas), según las tareas (procesos) que puede realizar simultáneamente. Los sistemas operativos más populares son: Windows 95, Windows NT, Windows 2000, Windows XP, UNIX, Linux... El único sistema operativo, prácticamente gratis, es Lincex, que cada día tiene mayor aceptación.

Un sistema operativo permite utilizar el hardware y comunicarse con el resto del software del sistema. El sistema operativo consta de varios componentes importantes. *Kernel* del sistema, componente central del sistema; *sistema de administración de memoria*, asigna un área de memoria para cada programa que se está ejecutando; *administrador del sistema de archivos*, que organiza y controla el uso de los discos; *controladores de dispositivos*, que controlan los dispositivos de hardware conectados a la computadora, y *bibliotecas del sistema*, que contiene todos los tipos de programas de utilidad que se pueden llamar por programas de usuario.

Un programa es un conjunto de instrucciones que controlan (dirigen) a una computadora; más formalmente, *un programa de computadora* es un conjunto de instrucciones internas utilizadas para ejecutarse en una computadora y que produzca un resultado concreto. Otro término para un programa o conjunto de programas es *software*; ambos términos se utilizan indistintamente. El proceso de escribir un programa, o *software*, se denomina **programación** y el conjunto de instrucciones que se pueden utilizar para construir un programa se denomina *lenguaje de programación*. Así pues, los lenguajes utilizados para escribir programas de computadoras son los *lenguajes de programación* y *programadores* son los escritores y diseñadores de programas.

Cuando un programa se **ejecuta** (*run*) normalmente hay dos tipos de entrada a una computadora. El **programa** es un tipo de entrada (contiene instrucciones que la computadora debe realizar y seguir). El otro tipo de entrada son los **datos** para el programa. Es la información que procesará la computadora. Por ejemplo, si es un programa de corrección de textos, los datos como el propio programa son entradas. La salida es el resultado (o resultados) producido cuando la computadora sigue las instrucciones del programa. Si el programa es el de corrección de la sintaxis de un texto, la salida del programa es una lista de palabras que están mal escritas (tienen errores de ortografía). Cuando se proporciona a la computadora un programa y datos y se dice a la computadora que siga las instrucciones del programa, se realiza una operación que se denomina **ejecución** (*running*) del programa con los datos y el computador se dice que ejecuta el programa con los datos.

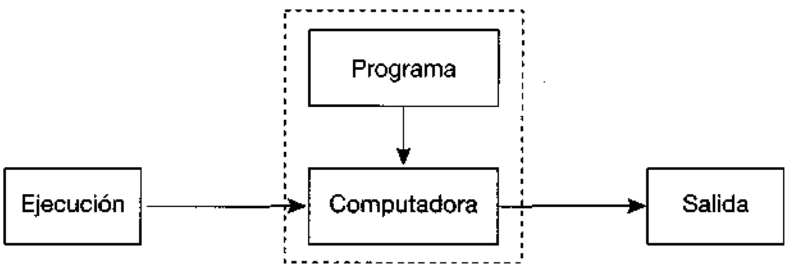


Figura 1.14. Ejecución de un programa.

Software

La palabra *software* significa simplemente programas. Por consiguiente, una compañía de software es una compañía que produce o «fabrica» programas. El software en su computadora es la colección de programas de su computadora.

1.6. LOS LENGUAJES DE PROGRAMACIÓN

Los *lenguajes de programación* sirven para escribir programas que permitan la comunicación usuario/máquina. Unos programas especiales llamados *traductores* (**compiladores** o **intérpretes**) convierten las instrucciones escritas en lenguajes de programación en instrucciones escritas en lenguajes máquina (0 y 1, *bits*) que ésta pueda entender.

Los *programas de utilidad*[8] facilitan el uso de la computadora. Un buen ejemplo es un *editor de textos* que permite la escritura y edición de documentos. Este libro ha sido escrito en un editor de textos o *procesador de palabras* («***word procesor***»).

Los programas que realizan tareas concretas, nóminas, contabilidad, análisis estadístico, etc., se denominan *programas de aplicación.* A lo largo del libro se verán pequeños programas de aplicación que muestran los principios de una buena programación de computadora (véase la Figura 1.15).

Figura 1.15. Relación entre programas de aplicación y programas del sistema.

Se debe diferenciar entre el acto de crear un programa y la acción de la computadora cuando ejecuta las instrucciones del programa. La creación de un programa se hace inicialmente en papel y a continuación se introduce en la computadora y se convierte en lenguaje entendible por la computadora.

El propósito de un lenguaje informático (lenguaje de computadora) es permitir a las personas comunicarse con una computadora. Los lenguajes de los humanos y los lenguajes de la máquina son muy diferentes, ya que las características y posibilidades de las personas y de las máquinas son muy diferentes. Los lenguajes de computadoras permiten a las personas escribir en un lenguaje que sea más apropiado a las características humanas y se puedan traducir al lenguaje máquina de diferentes tipos de máquinas.

Los principales tipos de lenguajes utilizados en la actualidad son tres:

- *lenguajes máquina,*
- *lenguaje de bajo nivel (ensamblador),*
- *lenguajes de alto nivel.*

[8] *Utility*: programa de utilidad.

1.6.1. Instrucciones a la computadora

Los diferentes pasos (*acciones*) de un algoritmo se expresan en los programas como *instrucciones, sentencias* o *proposiciones* (normalmente el término **instrucción** se suele referir a los lenguajes máquina y bajo nivel, reservando la **sentencia** o **proposición** para los lenguajes de alto nivel). Por consiguiente, un programa consta de una secuencia de instrucciones, cada una de las cuales especifica ciertas operaciones que debe ejecutar la computadora.

La elaboración de un programa requerirá conocer el juego o repertorio de instrucciones del lenguaje. Aunque en el Capítulo 3 se analizarán con más detalle las instrucciones, adelantaremos los tipos fundamentales de instrucciones que una computadora es capaz de manipular y ejecutar. Las instrucciones básicas y comunes a casi todos los lenguajes de programación se pueden condensar en cuatro grupos:

- *Instrucciones de entrada/salida.* Instrucciones de transferencia de información y datos entre dispositivos periféricos (teclado, impresora, unidad de disco, etc.) y la memoria central.
- *Instrucciones aritmético-lógicas.* Instrucciones que ejecutan operaciones aritméticas (suma, resta, multiplicación, división, potenciación), lógicas (operaciones `and`, `or` `not`, etc.).
- *Instrucciones selectivas.* Instrucciones que permiten la selección de tareas alternativas en función de los resultados de diferentes expresiones condicionales.
- *Instrucciones repetitivas.* Instrucciones que permiten la repetición de secuencias de instrucciones un número determinado de veces.

1.6.2. Lenguajes máquina

Los **lenguajes máquina** son aquellos que están escritos en lenguajes directamente inteligibles por la máquina (computadora), ya que sus instrucciones son *cadenas binarias* (cadenas o series de caracteres —dígitos— 0 y 1) que especifican una operación, y las posiciones (dirección) de memoria implicadas en la operación se denominan *instrucciones de máquina* o *código máquina*). El código máquina es el conocido código binario.

Las instrucciones en lenguaje máquina dependen del *hardware* de la computadora y, por tanto, diferirán de una computadora a otra. El lenguaje máquina de un PC (computadora personal) será diferente de un sistema HP (Hewlett Packard), Dell, Compaq o un sistema de IBM.

Las *ventajas* de programar en lenguaje máquina se refieren, fundamentalmente, a la posibilidad de cargar (transferir un programa a la memoria) sin necesidad de traducción posterior, lo que supone una velocidad de ejecución superior a cualquier otro lenguaje de programación.

Los *inconvenientes* —en la actualidad— superan a las ventajas, lo que hace prácticamente no recomendables los lenguajes máquina al programador de aplicaciones. Estos inconvenientes son:

- dificultad y lentitud en la codificación,
- poca fiabilidad,
- dificultad grande de verificar y póner a punto los programas,
- los programas sólo son ejecutables en el mismo procesador (UPC, *Unidad Central de Proceso*).

Para evitar los lenguajes máquina, desde el punto de vista del usuario, se han creado otros lenguajes que permiten escribir programas con instrucciones similares al lenguaje humano (por desgracia casi siempre inglés, aunque existen raras excepciones, como es el caso de las antiguas versiones españolas del lenguaje LOGO). Estos lenguajes se denominan de *alto* y *bajo nivel.*

1.6.3. Lenguajes de bajo nivel

Los **lenguajes de bajo nivel** son más fáciles de utilizar que los lenguajes máquina, pero, al igual que ellos, dependen de la máquina en particular. El lenguaje de bajo nivel por excelencia es el *ensambla-*

dor (*assembly languaje*). Las instrucciones en lenguaje ensamblador son instrucciones conocidas como ***nemotécnicos*** (*mnemonics*). Por ejemplo, nemotécnicos típicos de operaciones aritméticas son: en inglés, `ADD`, `SUB`, `DIV`, etc.; en español, `SUM`, `RES`, `DIV`, etc.

Una instrucción típica de suma sería:

```
ADD, M, N, P
```

Esta instrucción podía significar «*sumar el número contenido en la posición de memoria M al número almacenado en la posición de memoria N y situar el resultado en la posición de memoria P*». Evidentemente, es mucho más sencillo recordar la instrucción anterior con un nemotécnico que su equivalente en código máquina:

```
0110    1001    1010    1011
```

Un programa escrito en lenguaje ensamblador no puede ser ejecutado directamente por la computadora —en esto se diferencia esencialmente del lenguaje máquina—, sino que requiere una fase de *traducción* al lenguaje máquina.

El programa original escrito en lenguaje ensamblador se denomina *programa fuente* y el programa traducido en lenguaje máquina se conoce como *programa objeto*, ya directamente inteligible por la computadora.

El traductor de programas fuente a objeto es un programa llamado *ensamblador* (*assembler*), existente en casi todas las computadoras (Fig. 1.16).

No se debe confundir —aunque en español adoptan el mismo nombre— *el programa ensamblador (assembler)*, encargado de efectuar la traducción del programa fuente escrito a lenguaje máquina,

Figura 1.16. Programa ensamblador.

con el *lenguaje ensamblador (assembly languaje)*, lenguaje de programación con una estructura y gramática definidas.

Los lenguajes ensambladores presentan la *ventaja* frente a los lenguajes máquina de su mayor facilidad de codificación y, en general, su velocidad de cálculo.

Los *inconvenientes* más notables de los lenguajes ensambladores son:

- Dependencia total de la máquina, lo que impide la transportabilidad de los programas (posibilidad de ejecutar un programa en diferentes máquinas). El lenguaje ensamblador del PC es distinto del lenguaje ensamblador del Apple Macintosh.
- La formación de los programadores es más compleja que la correspondiente a los programadores de alto nivel, ya que exige no sólo las técnicas de programación, sino también el conocimiento del interior de la máquina.

Hoy día los lenguajes ensambladores tiene sus aplicaciones muy reducidas en la programación de aplicaciones y se centran en aplicaciones de tiempo real, control de procesos y de dispositivos electrónicos, etc.

1.6.4 Lenguajes de alto nivel

Los *lenguajes de alto nivel* son los más utilizados por los programadores. Están diseñados para que las personas escriban y entiendan los programas de un modo mucho más fácil que los lenguajes máquina y ensambladores. Otra razón es que un programa escrito en lenguaje de alto nivel es independiente de la máquina; esto es, las instrucciones del programa de la computadora no dependen del diseño del *hardware* o de una computadora en particular. En consecuencia, los programas escritos en lenguaje de alto nivel son *portables* o *transportables*, lo que significa la posibilidad de poder ser ejecutados con poca o ninguna modificación en diferentes tipos de computadoras; al contrario que los programas en lenguaje máquina o ensamblador, que sólo se pueden ejecutar en un determinado tipo de computadora. Los lenguajes de alto nivel presentan las siguientes *ventajas*:

- El tiempo de formación de los programadores es relativamente corto comparado con otros lenguajes.
- La escritura de programas se basa en reglas sintácticas similares a los lenguajes humanos. Nombres de las instrucciones, pueden ser `READ`, `WRITE`, `PRINT`, `OPEN`, etc.
- Las modificaciones y puestas a punto de los programas son más fáciles.
- Reducción del coste de los programas.
- Transportabilidad.

Los *inconvenientes* se concretan en:

- Incremento del tiempo de puesta a punto, al necesitarse diferentes traducciones del programa fuente para conseguir el programa definitivo.
- No se aprovechan los recursos internos de la máquina, que se explotan mucho mejor en lenguajes máquina y ensambladores.
- Aumento de la ocupación de memoria.
- El tiempo de ejecución de los programas es mucho mayor.

Al igual que sucede con los lenguajes ensambladores, los programas fuente tienen que ser traducidos por los programas traductores, llamados en este caso *compiladores* e *intérpretes*.

Los lenguajes de programación de alto nivel existentes hoy son muy numerosos, aunque la práctica demuestra que su uso mayoritario se reduce a

C ***C++*** ***COBOL*** ***FORTRAN*** ***Pascal*** ***Visual BASIC*** ***Java*** ***C#***

están muy extendidos:

Ada-95 ***Modula-2*** ***Prolog*** ***LISP*** ***Smalltalk*** ***Eiffel***

son de gran uso en el mundo profesional

Borland Delphi ***SQL*** ***Power Builder***

aunque hoy día el mundo Internet consume gran cantidad de recursos en forma de lenguajes de programación, tales como **Java**, **HTML**, **XML**, **JavaScript**, **Visual J** y últimamente **C#** y **PHP**.

1.7. TRADUCTORES DE LENGUAJE

Los *traductores de lenguaje* son programas que traducen a su vez los programas fuente escritos en lenguajes de alto nivel a código máquina. Los traductores se dividen en:

- *compiladores,*
- *intérpretes.*

1.7.1. Intérpretes

Un *intérprete* es un traductor que toma un programa fuente, lo traduce y a continuación lo ejecuta. Los programas intérpretes clásicos, como BASIC, prácticamente ya no se utilizan, aunque puede encontrar una vieja computadora que funcione con la versión QBasic bajo el Sistema Operativo DOS que corre en las computadoras personales. Sin embargo, está muy extendida la versión interpretada del lenguaje Smalltalk, un lenguaje orientado a objetos puro (véanse las Figuras 1.17 y 1.18). Los intérpretes han vuelto a renacer con la aparición de Java, ya que para entender el código en bytes (*bytecode*) al que traduce un compilador Java se necesita un intérprete.

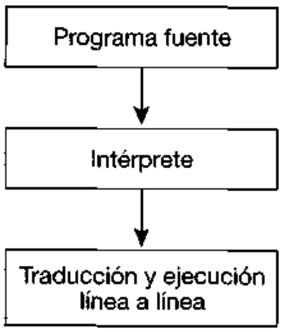


Figura 1.17. Intérprete.

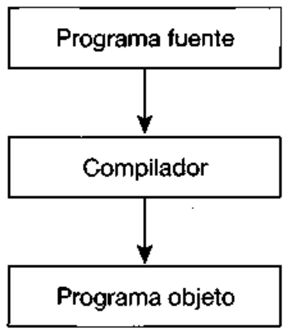


Figura 1.18. La compilación de programas.

1. Los datos son la entrada del programa.
2. La salida es la salida del programa.

Software

La palabra software significa simplemente programas. Por consiguiente, una compañía de software es una compañía que produce o «fabrica» programas. El software en su computadora es la colección de programas de su computadora.

1.7.2. Compiladores

Después que se ha diseñado el algoritmo y escrito el programa en un papel, se debe comenzar el proceso de introducir el programa en un archivo (fichero) en el disco duro de la computadora. La introducción y modificación de su programa en un archivo se hace utilizando un *editor de texto* o simplemente un *editor*, un programa que viene con su computadora, normalmente, y que le permite el almacenamiento y recuperación de lo que se ha escrito en el disco. El aprendizaje de cómo utilizar un editor hace la tarea de introducir un programa una tarea muy fácil.

El programa que se ha introducido está escrito en C o en Java, pero ni C ni Java son lenguajes máquina de su computadora, muy al contrario son lenguajes de alto nivel diseñados para hacer más fácil la programación que utilizando el lenguaje máquina. Una computadora no entiende los lenguajes de alto nivel. En consecuencia, un programa escrito en un lenguaje de alto nivel debe ser traducido a un lenguaje que la máquina pueda comprender. Los lenguajes que la computadora puede comprender (más o menos directamente) se llaman **lenguajes de bajo nivel**. La traducción de un programa escrito en un lenguaje de alto nivel, como C++ o Java, a un lenguaje que pueda entender la computadora se hace mediante otro programa conocido como **compilador**.

Los lenguajes de bajo nivel que la computadora puede entender directamente se conocen como lenguajes ensamblador o lenguajes máquina. En realidad, aunque son muy similares y en ocasiones se les considera sinónimos, tienen algunas pequeñas diferencias. El lenguaje que la computadora puede comprender directamente se denomina lenguaje máquina. El lenguaje ensamblador es casi la misma cosa, pero necesita un paso adicional para que la traducción pueda ser entendida por la máquina. Si un compilador traduce su programa de alto nivel a algún lenguaje de bajo nivel, no es exactamente lenguaje de máquina, se necesitará, por tanto, una pequeña traducción adicional antes de ser ejecutado en la computadora, pero normalmente este proceso suele ser automático y no es problemático. Veamos en la práctica cómo se ejecuta el programa por compilador.

Cuando se ejecuta un programa en lenguaje de alto nivel, tal como C, C++ o Java, se está ejecutando realmente una traducción de ese programa a un lenguaje de bajo nivel. Por consiguiente, antes de que se ejecute un programa escrito en un lenguaje de alto nivel, se debe ejecutar en primer lugar el compilador en el programa. Cuando se ejecuta un compilador sobre sus programa se dice que se **compila** el programa.

Una desventaja del proceso de traducción que se acaba de describir es que se necesita un compilador diferente para cada tipo de computadora y cada sistema operativo. Si se desea ejecutar su programa en lenguaje de alto nivel en tres máquinas diferentes, se necesitarán utilizar tres diferentes compiladores y se debe compilar su programa tres veces. Incluso si un fabricante construye una nueva computadora el fabricante deberá contratar a alguien para que le escriba un nuevo compilador para esa computadora. Esto es un problema, ya que los compiladores son programas muy grandes y caros, que llevan mucho tiempo fabricarlos. A pesar de los costes, este es el método en que trabajan la mayoría de los compiladores de lenguajes de alto nivel, C, C++,... Java, sin embargo, no utiliza ese método sino que utiliza un enfoque ligeramente diferente y más versátil para compilar.

Suele haber confusión en la terminología, debido a que un compilador, la entrada al programa compilador y la salida del programa compilador son también programas. Para evitar la confusión, el programa escrito en un lenguaje de programación (C++ o Java, si tuviéramos un compilador de pseudocódigo, el lenguaje algorítmico que explicaremos más adelante, en nuestro caso UPSAM) se denomina programa fuente o código fuente. El programa traducido a lenguaje de bajo nivel producido por el compilador se denomina normalmente el programa objeto o código objeto. La palabra código significa simplemente eso: un programa o una parte de programa. El archivo objeto se puede cargar en la memoria principal de la computadora y ejecutado por UCP.

Recuerde

El programa escrito en un lenguaje de alto nivel se introduce en la máquina con el editor y se llama código fuente; el compilador lo traduce en leguaje máquina y almacena el resultado en otro archivo llamado código objeto.
El código objeto se puede cargar en la memoria principal de la computadora y se ejecuta por la UCP.

Nota

Un compilador es un programa que traduce un programa en lenguaje de alto nivel, tal como un programa C / C++ / Java, en un programa de un lenguaje más sencillo que la computadora puede comprender más o menos directamente.

En resumen, para crear y ejecutar un programa en un lenguaje tal como C, C++ o Java, se necesita introducir el programa utilizando un editor de texto; ejecutar un programa especial denominado compilador, traducir el lenguaje de alto nivel a lenguaje máquina y a continuación cargar el lenguaje máquina en memoria principal y ejecutarlo.

La compilación y sus fases

La *compilación* es el proceso de traducción de programas fuente a programas objeto. El programa objeto obtenido de la compilación ha sido traducido normalmente a código máquina.

Para conseguir el programa máquina real se debe utilizar un programa llamado *montador* o *enlazador* (*linker*). El proceso de montaje conduce a un programa en lenguaje máquina directamente ejecutable (Fig. 1.20).

El proceso de ejecución de un programa escrito en un lenguaje de programación (por ejemplo, C) y mediante un compilador suele tener los siguientes pasos:

1. Escritura del *programa fuente* con un *editor* (programa que permite a una computadora actuar de modo similar a una máquina de escribir electrónica) y guardarlo en un dispositivo de almacenamiento (por ejemplo, un disco).
2. *Introducir* el programa fuente en memoria.
3. *Compilar* el programa con el compilador C.
4. *Verificar* y *corregir errores de compilación* (listado de errores).
5. Obtención del programa *objeto*.
6. El enlazador (*linker*) obtiene el *programa ejecutable*.
7. Se ejecuta el programa y, si no existen errores, se tendrá la salida del programa.

El proceso de ejecución sería el mostrado en las Figuras 1.20 y 1.21.

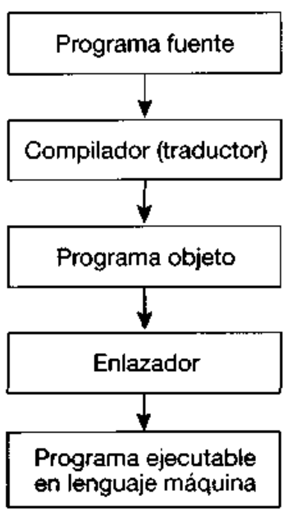


Figura 1.19. Fases de la compilación.

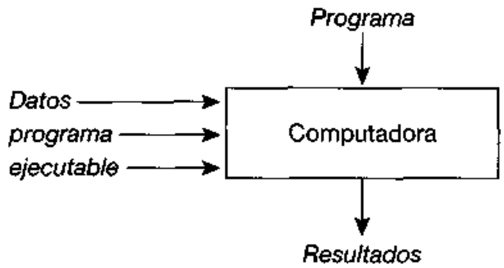


Figura 1.20. Ejecución de un programa.

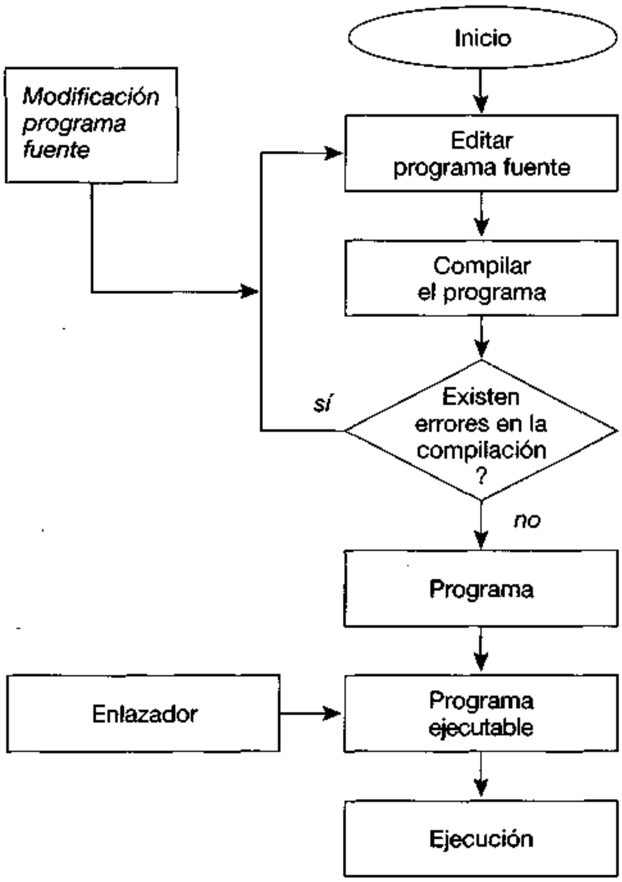


Figura 1.21. Fases de ejecución de un programa.

1.8. HISTORIA DE LOS LENGUAJES DE PROGRAMACIÓN

La historia de la informática y de la computación se ha caracterizado fundamentalmente en la existencia de centenares de lenguajes de programación, de forma tal que siempre se ha hablado de «la Babel de los lenguajes», aunque bien es cierto que son sólo unas decenas escasas los que han tenido un impacto significativo, y unos pocos los utilizados en la industria. los negocios y la ciencia.

Muchos de los lenguajes de programación actuales tienen sus raíces en los lenguajes que nacieron a finales de los cincuenta y primeros años de los sesenta, tales como *COBOL* (1960), *FORTRAN IV*

(1961), *BASIC* (1964), *LOGO* (1967). Estos lenguajes representaron la primera alternativa a los lenguajes ensambladores. En la década de los setenta y primeros años de los ochenta emergieron nuevos lenguajes, como *Pascal* (1971), *C* (1972) *y Ada* (1979), que se convirtieron en dominantes, desplazando, por ejemplo, a otros como ALGOL y BASIC. Otros lenguajes, como COBOL y FORTRAN, lograron adaptarse y se mantuvieron como lenguajes de referencia en el mundo de los negocios y en el campo científico, respectivamente, con versiones actualizadas y estandarizadas por organizaciones, como ANSI.

Todos estos lenguajes anteriores seguían el estilo de programación estructurada y se conocían *lenguajes de programación imperativos o estructurados.* En paralelo con el desarrollo de estos lenguajes surgieron dos nuevos estilos o paradigmas de programación; *programación funcional* (de los que *Haskell* es un ejemplo moderno) y *programación orientada a objetos. Smalltalk* y *SIMULA*, lenguajes antiguos por otra parte, se apoyaron en el concepto de clase como elemento fundamental de un programa. Posteriormente, a principios de los ochenta apareció **C++** como lenguaje imperativo con propiedades de orientación a objetos y que durante la década de los noventa ha sido el lengüaje de programación orientada a objetos por excelencia. En 1995 se presentó oficialmente por el constructor Sun el lenguaje **Java**, un lenguaje orientado a objetos y con una funcionalidad muy dirigida a la red Internet. En el año 2000 Microsoft presentó C#.

1.8.1. El lenguaje C: historia y características

C es el lenguaje de programación de propósito general asociado, de modo universal, al sistema operativo UNIX. Sin embargo, la popularidad, eficacia y potencia de C se ha producido porque este lenguaje no está prácticamente asociado a ningún sistema operativo, ni a ninguna máquina en especial. Esta es la razón fundamental por la cual C es conocido como el ***lenguaje de programación de sistemas por excelencia***.

C es una evolución de los lenguajes BCPL —desarrollado por Martin Richards— y B —desarrollado por Ken Thompson en 1970— para el primitivo UNIX de la computadora DEC PDP-7.

C nació realmente en 1978, con la publicación de *The C Programming Languaje*, por Brian Kernighan y Dennis Ritchie (Prentice Hall, 1978). Desde su nacimiento, C fue creciendo en popularidad y los sucesivos cambios en el lenguaje a lo largo de los años junto a la creación de compiladores por grupos no involucrados en su diseño hicieron necesario pensar en la estandarización de la definición del lenguaje C.

Así, en 1983, el American National Standard Institute (ANSI), una organización internacional de estandarización, creó un comité (el denominado X3J11) cuya tarea fundamental consistía en hacer *«una definición no ambigua del lenguaje C, e independiente de la máquina»*. Había nacido el estándar ANSI del lenguaje C. Con esta definición de C se asegura que cualquier fabricante de software que vende un compilador ANSI C incorpora todas las características del lenguaje especificadas por el estándar. Esto significa también que los programadores que escriban programas en C estándar tendrán la seguridad de que correrán sus modificaciones en cualquier sistema que tenga un compilador C.

C es un *lenguaje de alto nivel*, que permite programar con instrucciones de lenguaje de propósito general. También C se define como un lenguaje de programación estructurado de propósito general; aunque en su diseño también primó el hecho de fuera especificado como un lenguaje de programación de Sistemas, lo que proporciona una enorme cantidad de potencia y flexibilidad.

El estándar ANSI C formaliza construcciones no propuestas en la primera versión de C, en especial asignación de estructuras y enumeraciones. Entre otras aportaciones, se definió esencialmente una nueva forma de declaración de funciones (prototipos). Pero es esencialmente la biblioteca estándar de funciones otra de las grandes aportaciones.

Hoy, en el siglo XXI, C sigue siendo uno de los lenguajes de programación más utilizados en la industria del software, así como en institutos tecnológicos, escuelas de ingeniería y universidades. Prácticamente todos los fabricantes de sistemas operativos, UNIX, Linux, MacOS, Solaris..., soportan diferentes tipos de compiladores de lenguaje C.

Ventajas de C

El lenguaje C tiene una gran cantidad de ventajas sobre otros lenguajes y constituyen precisamente la razón fundamental de que después de casi dos décadas de uso C siga siendo uno de los lenguajes más populares, utilizados en empresas, organizaciones y fábricas de software de todo el mundo.

Algunas ventajas que justifican el uso todavía creciente del lenguaje C en la programación de computadoras son:

- El lenguaje C es poderoso y flexible, con órdenes, operaciones y funciones de biblioteca que se pueden utilizar para escribir la mayoría de los programas que corren en la computadora.
- C se utiliza por programadores profesionales para desarrollar software en la mayoría de los modernos sistemas de computadora.
- Se puede utilizar C para desarrollar sistemas operativos, compiladores, sistemas de tiempo real y aplicaciones de comunicaciones.
- Un programa C puede ser escrito para un tipo de computadora y trasladarse a otra computadora con pocas o ninguna modificación —propiedad conocida como *portabilidad*—. El hecho de que C sea **portable** es importante, ya que la mayoría de los modernos computadores tienen un compilador C. Una vez que se aprende C no tiene que aprenderse un nuevo lenguaje cuando se escriba un programa para otro tipo de computadora. No es necesario *reescribir* un problema para ejecutarse en otra computadora.

C se caracteriza por su velocidad de ejecución. En los primeros días de la informática los problemas de tiempo de ejecución se resolvían escribiendo todo o parte de una aplicación en lenguaje ensamblador (lenguaje muy cercano al lenguaje máquina). Debido a que existen muchos programas escritos en C, se han creado numerosas bibliotecas C para programadores profesionales que soportan gran variedad de aplicaciones. Existen bibliotecas del lenguaje C que soportan aplicaciones de bases de datos, gráficos, edición de texto, comunicaciones, etc.

Características técnicas de C

Hay numerosas características que diferencian a C de otros lenguajes y lo hacen eficiente y potente a la vez.

- Una nueva sintaxis para declarar funciones. Una declaración de función puede añadir una descripción de los argumentos de la función. Esta información adicional sirve para que los compiladores detecten más fácilmente los errores causados por argumentos que no coinciden.
- Asignación de estructuras (registros) y enumeraciones.
- *Preprocesador* más sofisticado.
- Una nueva definición de la biblioteca que acompaña a C. Entre otras funciones se incluyen: acceso al sistema operativo (por ejemplo, lectura y escritura de archivos), entrada y salida con formato, asignación dinámica de memoria, manejo de cadenas de caracteres.
- Una colección de cabeceras estándar que proporciona acceso uniforme a las declaraciones de funciones y tipos de datos.

Versiones actuales de C

En la actualidad son muchos los fabricantes de compiladores C, aunque los más populares entre los fabricantes de software son: Microsoft, Borland, HP, ...

Una evolución de C es el lenguaje **C++** (C con clases) que contiene, entre otras, todas las características de ANSI C. Los compiladores más empleados Visual C++ de Microsoft, Builder C++ de Borland, C++ bajo UNIX y Linux.

En el verano del 2000 Microsoft presentó y patentó una nueva versión de C++ que es C#, una evolución del C++ estándar.

1.8.2. Una breve historia de C++

C++ es heredero directo del lenguaje C que a su vez se deriva del lenguaje B [Richards, 1980]. C se mantiene como un subconjunto de C++. Otra fuente de inspiración de C++, como señala su autor Bjarne Stroustrup fue Simula 67, del que tomó el concepto de clase (con clases derivadas y funciones virtuales).

El lenguaje de programación C fue desarrollado por **Dennis Ritche** de AT&T Bell Laboratories que se utilizó para escribir y mantener el sistema operativo UNIX (hasta que apareció C, el sistema operativo UNIX fue desarrollado por **Ken Thompson** en AT&T Bell Laboratories mediante el lenguaje ensamblador o el lenguaje B). C es un lenguaje de propósito general que se puede utilizar para escribir cualquier tipo de programa, pero su éxito y popularidad está especialmente relacionado con el sistema operativo UNIX. Fue desarrollado como *lenguaje de programación de sistemas*; es decir, un lenguaje de programación para escribir *sistemas operativos* y utilidades (programas) del sistema. Los sistemas operativos son los programas que gestionan (administran) los *recursos de la computadora*. Ejemplos bien conocidos de sistemas operativos además de UNIX son los ya citados MS/DOS, OS/2, MVS, Lynux, Windows 95/98, Windows NT, Windows 2000, OS Mac, etc.

La especificación formal del lenguaje C es un documento escrito por Ritchie titulado *The C Reference Manual*. En 1997 **Ritchie** y **Brian Kernighan** ampliaron ese documento y publicaron un libro referencia del lenguaje *The C Programming Language* (también conocido por el **K&R**).

Aunque C es un lenguaje muy potente, tiene dos características que lo hacen inapropiado como una introducción moderna a la programación. Primero, C requiere un nivel de sofisticación a sus usuarios que les obliga a un difícil aprendizaje para los programadores principiantes, ya que es de comprensión difícil. Segundo, C fue diseñado al principio de los setenta, y la naturaleza de la programación ha cambiado de modo significativo en la década de los ochenta y noventa.

Para subsanar estas «deficiencias» Bjarne Stroustrup de AT&T Bell Laboratories desarrolló C++ al principio de la década de los ochenta. Stroustrup diseñó **C++ como un mejor C**. En general, C estándar es un subconjunto de C++ y la mayoría de los programas C son también programas C++ (la afirmación inversa no es verdadera). C++, además de añadir propiedades a C, presenta características y propiedades de *programación orientada a objetos*, que es una técnica de programación muy potente y que se verá en la última parte de este libro.

Se han presentado varias versiones de C++ y su evolución continúa en al actualidad. Las características más notables que han ido incorporándose a C++ son: herencia múltiple, genericidad, plantillas, funciones virtuales, excepciones, etc. C++ ha ido evolucionando año a año y como su autor ha explicado: «*evolucionó siempre para resolver problemas encontrados por los usuarios y como consecuencia de conversaciones entre el autor, sus amigos y sus colegas*»[9].

C++ comenzó su proyecto de estandarización ante el comité ANSI y su primera referencia es *The Annotated C++ Reference Manual*[10]. En diciembre de 1989 se reunió el comité X3J16 del ANSI por iniciativa de Hewlett Packard. En junio de 1991 la estandarización de ANSI pasó a formar parte de un esfuerzo de estandarización ISO.

En 1995 se publicó un borrador de estándar para su examen y en 1998 se aprobó el estándar C++ internacional. Stroustrup publicó en 1998 la tercera edición de su libro *The C++ Programming Language* y en el año 2000 una edición especial. Este libro sigue el estándar ANSI/ISO C++.

1.8.3. Breve historia de Java

Ocho años después de su lanzamiento, **Java** se ha convertido en un estándar de la industria, en un lenguaje de programación para desarrollo de aplicaciones tanto de propósito general como de Internet y

[9] Stroustrup 98, *The C++ Programmig Languaje*, Addison-Wesley, 1998, pág. 12. Una edición especial publicada en el año 2000 ha sido traducida por un equipo de profesores de la Universidad Pontificia de Salamanca dirigida por el autor de esta obra.

[10] Existe versión española de Addison-Wesley Díaz de Santos y traducida por los profesores Manuel Katrib y Luis Joyanes.

también en un lenguaje para comenzar la formación en programación, al tener características excelentes para el aprendizaje.

Java, desarrollado por Sun Microsystems en 1995, es un magnífico y completo lenguaje de programación orientado a objetos diseñado para distribuir contenidos a través de una red. Una de sus principales características es que permite operar de forma independiente de la plataforma y del sistema operativo que se esté utilizando. Esto quiere decir que permite crear una aplicación que podrá descargarse de la red y funcionar posteriormente en cualquier tipo de plataforma de hardware o software. Generalmente, por el contrario, todo programa o aplicación queda atado a dos cosas: al *hardware* y al sistema operativo. Así, por ejemplo, una aplicación Windows sólo funcionará en plataforma Wintel (equipada con procesadores Intel y sistema operativo Windows) igual que una versión creada para Mac sólo funciona sobre Power PC, Imac o Mac OS; o la misma aplicación desarrollada para Unix, sólo lo hace sobre plataformas Unix y no hay forma de que corra sobre otra máquina.

La idea de Java, por el contrario, es poner una capa sobre cualquier plataforma de *hardware* y sobre cualquier sistema operativo para permitir que cualquier aplicación desarrollada en Java quede ligada únicamente a Java, independizada, por lo tanto, de la plataforma. Esta concepción queda recogida en el concepto de máquina virtual **JVM** (Java Virtual Machine), un software que interpreta instrucciones para cualquier máquina sobre la que esté corriendo y que permite, una vez instalado, que una misma aplicación pueda funcionar en un PC o en un Mac sin tener que tocarla. Hoy en día, cualquier sistema operativo moderno (Windows, Macintosh, Linux, Unix, Solaris, etc.) cuenta con una JVM. Así, lo que hace Java en combinación con esta «máquina» es funcionar como *hardware* y como sistema operativo virtual, emulando en software una CPU universal. Al instalar Java ésta actuará como una capa de abstracción entre un programa y el sistema operativo, otorgando una total independencia de lo que haya por debajo, es decir: cualquier aplicación funcionará en cualquier máquina e incluso en cualquier dispositivo.

Otra gran ventaja es que los programadores no tendrán que desarrollar varias versiones de la misma aplicación, puesto que el modelo de desarrollo es el mismo se trate del dispositivo más pequeño o del más grande de los servidores. Otra gran ventaja es que permite que todas las máquinas, plataformas y aplicaciones se comuniquen entre sí accediendo desde cualquier equipo, donde quiera que esté situado, a las aplicaciones que residan en una red, ya sea Internet o una intranet o extranet.

En definitiva, se puede decir que Java es lo más cercano a un lenguaje de computación universal que existe hoy día, lo que significa que puede correr en cualquier plataforma siempre y cuando una máquina virtual haya sido escrita para ella.

La genealogía de Java

Java es un descendiente de C++ que a su vez es descendiente directo de C. Muchas características de Java se han heredado de estos dos lenguajes. De C, Java ha heredado su sintaxis y de C++, las características fundamentales de programación orientada a objetos.

El diseño original de Java fue concebido por James Gosling, Patrick Naughton, Chris Warth, Ed Frank y Mike Sheridan, ingenieros y desarrolladores de Sun Microsystems en 1991 y tardaron dieciocho meses en terminar la primera versión de trabajo. Este lenguaje se llamó inicialmente, como se ha comentado, «Oak», y se le cambió el nombre por Java en la primavera de 1995.

Sorprendentemente, las inquietudes originales para la creación de «Oak» no era Internet. En realidad, se buscaba un lenguaje independiente de la plataforma (es decir, de arquitectura neutra) que se pudiera utilizar para crear software que se incrustara en dispositivos electrónicos diversos, tales como controles remotos u hornos de microondas. Aunque el modelo de lenguaje elegido fue C++ se empezaron a encontrar que si bien se podía compilar un programa C++ en cualquier tipo de CPU (Unidad Central de Proceso), se requería, sin embargo, un compilador C++ completo que corriese en esa CPU. El problema, en consecuencia, se convertía en compiladores caros y gran consumo de tiempo para crear los programas. Sobre esas premisas Gosling y sus colegas comenzaron a pensar en un lenguaje *portable,* independiente de la plataforma que se pudiera utilizar para producir código que corriera (se ejecutara) en una amplia variedad de CPUs y bajo diferentes entornos. Entonces comenzó a aparecer el nuevo proyecto y comenzó a llamarse Java.

¿Por qué Java es importante para Internet?

Internet ha ayudado considerablemente a «catapultar» a Java al *zenit* del mundo de la programación de computadoras y Java, a su vez, ha tenido un profundo impacto en Internet. La razón es muy simple: Java extiende el universo de los objetos que se mueven libremente en el ciberespacio que forma la red Internet. En una red existen dos grandes categorías de objetos que se transmiten entre las computadoras conectadas (el servidor y la computadora personal): información pasiva y dinámica, programas interactivos. Un ejemplo fácil de datos pasivos son los correos electrónicos que usted recibe en su computadora o una página *Web* que se baja de la red. Incluso si descarga un programa está recibiendo datos pasivos hasta tanto no ejecute dicho programa. Sin embargo, existen otros tipos de objetos que se transmite por la red: programas dinámicos autoejecutables que son agentes activos en la computadora cliente.

Estos programas en red dinámicos presentan serios problemas de seguridad y *portabilidad.* Java ha resuelto gran cantidad de problemas con un nuevo modelo de programa: el *applet.*

Java se puede utilizar para crear dos tipos de programas: aplicaciones y *applets.* Una *aplicación* es un programa que se ejecuta en su computadora bajo el sistema operativo de su computadora; en esencia, es un programa similar al creado utilizando C, C++ o Pascal. Cuando se utiliza para crear aplicaciones, Java es un lenguaje de propósito general similar a cualquier otro y con características que lo hacen idóneo para programación orientada a objetos. Java tiene una característica notable que no tienen otros lenguajes: la posibilidad de crear *applets.* Un *applet* es una aplicación diseñada para ser transmitida por Internet y ejecutada por un navegador Web compatible con Java. Un *applet* es realmente un pequeño programa Java, descargado dinámicamente por la red, tal como una imagen, un archivo de sonido, un archivo musical o un videoclip, pero con una notable propiedad: es un programa inteligente que puede reaccionar dinámicamente a entradas y cambios del usuario.

Java es un lenguaje idóneo para resolver los problemas de seguridad y portabilidad inherente a los sistemas que trabajan en red. La razón fundamental de este aserto reside en el hecho de que la salida de un compilador Java no es un código ejecutable sino códigos de bytes (*bytecode*). Un *bytecode* es un conjunto de instrucciones muy optimizadas diseñadas para ser ejecutadas por un sistema en tiempo de ejecución Java, denominado máquina virtual Java (*Java Virtual Machine, JVM*) que actúa como un intérprete para los *bytecodes.* La traducción de un programa en códigos de bytes facilita la ejecución del programa en una amplia variedad de entornos y plataformas. La razón es simple: sólo es preciso implementar JVM en cada plataforma.

1.8.4. C# el más joven

A pesar del éxito de Java, indudable por otra parte, también tiene sus lagunas. Entre ellas la dificultad para la *interoperabilidad de lenguajes cruzados o programación de lenguajes mezclados.* Es decir, la capacidad para que el código producido por un lenguaje funcione fácilmente con el código producido por otro. La interoperabilidad de lenguajes cruzados se requiere para la creación de sistemas de *software* grandes y distribuidos, así como para la construcción de componentes de *software.*

Otra carencia notable de Java es la no integración completa de la plataforma Java, aunque los programas Java se pueden ejecutar en un entorno Windows (suponiendo que se ha instalado la Máquina Virtual Java). Para responder a estas necesidades Microsoft creó C# dentro de su estrategia global .NET y nombró arquitecto jefe del proyecto a Anders Hejlsberg, uno de los grandes expertos mundiales en lenguajes de programación (entre otros proyectos creó en los ochenta el popular Turbo Pascal).

C# está relacionado directamente con C, C++ y Java. De C ha heredado su sintaxis, muchas de las palabras reservadas y sus operadores. De C++ ha heredado su modelo de objetos. La relación con Java es más bien de «amor/odio». C# y Java son familia, más bien primos hermanos, sin embargo C# no es descendiente de Java, aunque compartan unos descendientes comunes, C y C++. Por ejemplo, ambos soportan programación distribuida y ambos utilizan código intermedio para conseguir portabilidad, pero difieren en los detalles. C# añade importantes innovaciones al arte de la programación. Por ejemplo, C# incluye delegados, propiedades, indexadores y eventos como elementos del lenguaje.

También añade sintaxis que soporta atributos. Además, C# racionaliza la creación de componentes, eliminado los problemas asociados con COM. También, en este caso como Java, ofrece una cantidad significativa de verificación de errores en tiempo de ejecución, seguridad y gestión de la ejecución. En resumen, C# reune la potencia de C++ con la seguridad de tipos de Java.

C# representa en la primera década del siglo XXI a un lenguaje de programación con la mayoría de las grandes propiedades que todo buen lenguaje debe cumplir. Por esta razón es un lenguaje que ningún programador profesional puede ignorar si desea realizar desarrollos elegantes y eficientes.

ACTIVIDADES DE PROGRAMACIÓN RESUELTAS

1.1. *Indique las partes fundamentales de una computadora.*

En su configuración más básica una computadora está constituida por una Unidad Central de Procesamiento (UCP), una Memoria Principal y unas Unidades de Entrada y Salida o periféricos y las funciones desempeñadas por cada una de estas partes son las siguientes:

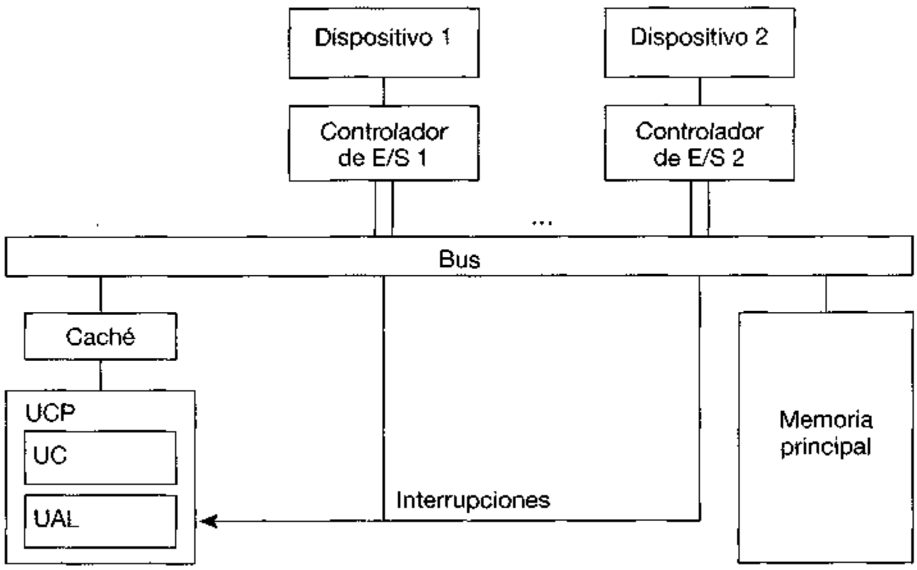
Las unidades de entrada/salida se encargan de los intercambios de información con el exterior.

La memoria principal almacena tanto las instrucciones que constituyen los programas como los datos y resultados obtenidos.

La UCP es la encargada de la ejecución de los programas almacenados en memoria. En la UCP destacan la Unidad de Control (UC) que se encarga de ejecutar las instrucciones y la Unidad Aritmético Lógica (UAL) que efectúa las operaciones.

1.2. *¿Qué es el bus del sistema de un PC?*

El bus del sistema es una ruta eléctrica de múltiples líneas —clasificables como líneas de direcciones, datos y control— que conecta la UCP, la memoria y los dispositivos de entrada/salida. Los dispositivos de entrada/salida se conectan al bus a través de controladores, que generan micro órdenes para gobernar a los periféricos.



1.3. *Defina los siguientes conceptos:*

UCP.
Programa.
Lenguaje de alto nivel.

Compilador.
Intérprete.
Lenguaje máquina.
Bit.

La UCP puede definirse como la parte activa del computador que contiene la UAL y la UC y puede sumar números, compararlos, activar dispositivos de entrada/salida, etc.

Un programa es un conjunto de instrucciones con una determinada sintaxis que pueden ser interpretadas y ejecutadas por la UCP.

Los lenguajes de programación de alto nivel son aquellos en que las instrucciones o sentencias a la computadora son escritas con palabras similares a los lenguajes humanos.

Un compilador es un programa que traduce un lenguaje de alto nivel al código máquina. Los compiladores traducen todo el código fuente y producen un programa ejecutable independiente.

Un intérprete traduce un programa escrito en un lenguaje de alto nivel a código máquina según se va ejecutando.

Los lenguajes máquina son aquellos cuyas instrucciones son directamente entendibles por la computadora y no necesitan traducción posterior para que la UCP pueda comprender y ejecutar el programa. Las instrucciones en lenguaje máquina son una secuencia de bits que especifican la operación y las celdas de memoria implicadas en una operación.

Un bit es un dígito numérico en base 2.

1.4. Rellene los espacios en blanco en las siguientes afirmaciones:

a) Las computadoras procesan datos controladas por series de instrucciones que se denominan..........
b) Los lenguajes de programación pueden dividirse en los siguientes tipos generales: de alto nivel.....
c) Un ensamblador traduce un programa en .... a código máquina.
d) Los programas escritos en lenguaje ensamblador se han de .... si se tienen que ejecutar en un computador de distinta arquitectura.
e) La memoria .... es la usada para almacenar los programas mientras se ejecutan, a diferencia de la memoria secundaria, que es memoria no volátil usada para almacenar programas y datos entre ejecuciones.

a) programas
b) ensamblador y lenguajes máquina
c) lenguaje ensamblador
d) reescribir
e) principal

1.5. Enumere y describa las principales jerarquías de memoria que conozca atendiendo a tiempo de acceso y capacidad:

Siguiendo un orden creciente en cuanto a tiempo de acceso y capacidad, lo primero que se debería mencionar son *los registros de procesador*. Estos registros tienen un tiempo de acceso muy pequeño y una capacidad mínima e interaccionan continuamente con la CPU, ya que forman parte de ella.

A continuación estarían las *memorias caché* que son memorias de pequeña capacidad y pequeño tiempo de acceso que, precisamente por ello, se colocan entre la CPU y la memoria central.

El siguiente nivel a considerar es la *memoria principal*, donde residen los programas y los datos. La UCP lee y escribe datos en ella, aunque con menos frecuencia que en los niveles anteriores. Tiene un tiempo de acceso bastante rápido y gran capacidad.

En último lugar estarían las *memorias secundarias*, como cintas magnéticas, discos magnéticos, discos compactos, etc., donde se almacenan los programas y los datos para que no se pierdan cuando se apaga el ordenador. Estas memorias también se usan para apoyo de la memoria central (memoria virtual) en el caso de que ésta sea insuficiente. Son memorias de gran capacidad pero cuyo tiempo de acceso es más lento.

REVISIÓN DEL CAPÍTULO

Conceptos clave

- Almacenamiento secundario.
- CD-RW.
- Compilación independiente.
- Compilador.
- Computadora.
- CPU.
- Disco.
- Disquete.
- DVD.
- DVD-RW.
- Ensamblador.
- Entrada/salida.
- Hardware.
- Intérprete.
- Lenguaje de alto nivel.
- Lenguaje de bajo nivel.
- Lenguaje de programación.
- Lenguaje ensamblador.
- Lenguaje máquina.
- Máquina virtual Java.
- Microprocesador
- Memoria auxiliar.
- Memoria central.
- Memoria.
- Microprocesador.
- Portabilidad.
- Procesador.
- Programa de computadora.
- RAM.
- ROM.
- Sistema operativo.
- Software de aplicaciones.
- Software del sistema.
- UCP.
- Unidad Central de Proceso.

Resumen

Una computadora es una máquina para procesar información y obtener resultados en función de unos datos de entrada.

Hardware: parte física de una computadora (dispositivos electrónicos).
Software: parte lógica de una computadora (programas).

Las computadoras se componen de:

- Dispositivos de Entrada/Salida (E/S).
- Unidad Central de Proceso (Unidad de Control y Unidad Lógica y Aritmética).
- Memoria central.
- Dispositivos de almacenamiento masivo de información (memoria auxiliar o externa).

El *software del sistema* comprende, entre otros, el *sistema operativo* (Windows 98/XP o Linux en computadoras personales) y los compiladores e intérpretes de lenguajes de programación.

Los *lenguajes de programación* de alto nivel están diseñados para hacer más fácil la escritura de programas que los lenguajes de bajo nivel. Existen numerosos lenguajes de programación, cada uno de los cuales tiene sus propias características y especialidades. Los programas escritos en lenguajes de alto nivel son, normalmente, más fáciles de transportar a máquinas diferentes que los escritos en lenguajes de bajo nivel.

Los programas escritos en lenguajes de alto nivel deben ser traducidos por un compilador antes de que se puedan ejecutar en una máquina específica. En la mayoría de los lenguajes de programación se requiere un compilador para cada máquina en la que se desea ejecutar programas escritos en un lenguaje específico. Los lenguajes de programación se clasifican en:

- *alto nivel*: Pascal, C, C++, C#, Java, Visual Basic, VB.NET, FORTRAN, Ada, Modula-2, Delphi, etc.
- *bajo nivel*: Ensamblador.
- *máquina*: Código máquina.

Los programas traductores de lenguajes son:

- *compiladores*,
- *intérpretes*.

EJERCICIOS

1.1. Definir los siguientes términos:

a) programa de computadora
b) programación
c) lenguaje de programación
d) lenguaje de alto nivel
e) lenguaje máquina
f) lenguaje ensamblador
g) compilador
h) intérprete

1.2. Enumerar los diferentes tipos de lenguajes de programación utilizados a lo largo de las diferentes generaciones de computadoras. ¿Cuáles tiene previsto utilizar en su aprendizaje? ¿Por qué?

1.3. Describa las características más sobresalientes de los lenguajes de programación Pascal, C, C++, Java y C#. ¿Cuál cree el mejor para la enseñanza de la programación a nivel de iniciación? ¿Por qué?

1.4. El almacenamiento de datos en dispositivos externos de memoria es una necesidad vital tanto durante el aprendizaje como durante su vida profesional. ¿Cuáles piensa utilizar en su aprendizaje en su casa y en su centro de enseñanza? Explique sus características y compárelas con las características de los otros dispositivos explicados en este capítulo. Si puede elegir cuáles cree que serán los más idóneos en el proceso de aprendizaje.

1.5. ¿Cómo influye el conocimiento del hardware en la formación del aprendizaje de la programación?

1.6. Explicar las características y los tipos de software del sistema y software de aplicaciones que conoce.

1.7. Enumere los sistemas operativos que conoce y explique las razones por las que va a usar uno en concreto.

1.8. ¿Por qué el procesador es una parte tan importante de una computadora?

1.9. ¿Qué funciones realiza la Unidad Central de Proceso?

1.10. Explique las misiones de una memoria RAM así como sus diferentes tipos. ¿Cuál tiene su computadora de trabajo? ¿Explique las diferencias entre las memorias RAM que conoce con indicación de sus ventajas e inconvenientes?

1.11. ¿Cuál sería la configuración ideal de una computadora para aprender a programar en la época en que usted lea este libro?

1.12. ¿Cuáles son los dispositivos de almacenamiento masivo más usuales del tipo CD-ROM, DVD?

2

RESOLUCIÓN DE PROBLEMAS CON COMPUTADORA Y HERRAMIENTAS DE PROGRAMACIÓN

CONTENIDO

2.1. Fases en la resolución de problemas.
2.2. Programación modular.
2.3. Programación estructurada.
2.4. Concepto y características de algoritmos.
2.5. Escritura de algoritmos.
ACTIVIDADES DE PROGRAMACIÓN RESUELTAS.
REVISIÓN DEL CAPÍTULO.
Conceptos clave.
Resumen.
EJERCICIOS.

Este capítulo le introduce a la metodología a seguir para la resolución de problemas con computadoras y con un lenguaje de programación como C.

La resolución de un problema con una computadora se hace escribiendo un programa, que exige al menos los siguientes pasos:

1. Definición o análisis del problema.
2. Diseño del algoritmo.
3. Transformación del algoritmo en un programa.
4. Ejecución y validación del programa.

Uno de los objetivos fundamentales de este libro es el *aprendizaje y diseño de los algoritmos*. Este capítulo introduce al lector en el concepto de algoritmo y de programa, así como las herramientas que permiten «dialogar» al usuario con la máquina: *los lenguajes de programación*.

2.1. FASES EN LA RESOLUCIÓN DE PROBLEMAS

El proceso de resolución de un problema con una computadora conduce a la escritura de un programa y a su ejecución en la misma. Aunque el proceso de diseñar programas es —esencialmente— un proceso creativo, se puede considerar una serie de fases o pasos comunes, que generalmente deben seguir todos los programadores.

Las fases de resolución de un problema con computadora son:

- Análisis del problema.
- Diseño del algoritmo.
- Codificación.
- Compilación y ejecución.
- Verificación.
- Depuración.
- Mantenimiento.
- Documentación.

Constituyen el ciclo de vida del software y las fases o etapas usuales son:

- ***Análisis.*** El problema se analiza teniendo presente la especificación de los requisitos dados por el cliente de la empresa o por la persona que encarga el programa.
- ***Diseño.*** Una vez analizado el problema, se diseña una solución que conducirá a un *algoritmo* que resuelva el problema.
- ***Codificación (implementación).*** La solución se escribe en la sintaxis del lenguaje de alto nivel (por ejemplo, C) y se obtiene un programa.
- ***Compilación, ejecución y verificación.*** El programa se ejecuta, se comprueba rigurosamente y se eliminan todos los errores (denominados «*bugs*», en inglés) que puedan aparecer.
- ***Depuración y mantenimiento.*** El programa se actualiza y modifica cada vez que sea necesario, de modo que se cumplan todas las necesidades de cambio de sus usuarios.
- ***Documentación.*** Escritura de las diferentes fases del ciclo de vida del software, esencialmente el análisis, diseño y codificación, unidos a manuales de usuario y de referencia, así como normas para el mantenimiento.

Las dos primeras fases conducen a un diseño detallado escrito en forma de algoritmo. Durante la tercera etapa *(codificación)* se *implementa*[1] el algoritmo en un código escrito en un lenguaje de programación, reflejando las ideas desarrolladas en las fases de análisis y diseño.

La fase de *compilación y ejecución* traduce y ejecuta el programa. En las fases de *verificación y depuración* el programador busca errores de las etapas anteriores y los elimina. Comprobará que mientras más tiempo se gaste en la fase de análisis y diseño menos se gastará en la depuración del programa. Por último, se debe realizar la *documentación del programa.*

Antes de conocer las tareas a realizar en cada fase, vamos a considerar el concepto y significado de la palabra **algoritmo.** La palabra *algoritmo* se deriva de la traducción al latín de la palabra Alkhôwarîzmi[2], nombre de un matemático y astrónomo árabe que escribió un tratado sobre manipulación de números y ecuaciones en el siglo IX. Un **algoritmo** es un método para resolver un problema mediante una serie de pasos precisos, definidos y finitos.

[1] En la penúltima edición (21.ª) del **DRAE** (Diccionario de la Real Academia Española) se aceptó el término *implementar:* (Informática) «Poner en funcionamiento, aplicar métodos, medidas, etc., para llevar algo a cabo».

[2] Escribió un tratado matemático famoso sobre manipulación de números y ecuaciones titulado *Kitab al-jabr w'almugabala.* La palabra álgebra se derivó, por su semejanza sonora, de *al-jabr.*

Características de un algoritmo

- *preciso* (indicar el orden de realización en cada paso),
- *definido* (si se sigue dos veces, obtiene el mismo resultado cada vez),
- *finito* (tiene fin; un número determinado de pasos).

Un algoritmo debe producir un resultado en un tiempo finito. Los métodos que utilizan algoritmos se denominan *métodos algorítmicos,* en oposición a los métodos que implican algún juicio o interpretación que se denominan *métodos heurísticos.* Los métodos algorítmicos se pueden *implementar* en computadoras; sin embargo, los procesos heurísticos no han sido convertidos fácilmente en las computadoras. En los últimos años las técnicas de inteligencia artificial han hecho posible la *implementación* del proceso heurístico en computadoras.

Ejemplos de algoritmos son: instrucciones para montar en una bicicleta, hacer una receta de cocina, obtener el máximo común divisor de dos números, etc. Los algoritmos se pueden expresar por *fórmulas, diagramas de flujo* o **N-S** y *pseudocódigos*. Esta última representación es la más utilizada en lenguajes estructurados como C.

2.1.1. Análisis del problema

La primera fase de la resolución de un problema con computadora es el *análisis del problema.* Esta fase requiere una clara definición, donde se contemple exactamente lo que debe hacer el programa y el resultado o solución deseada.

Dado que se busca una solución por computadora, se precisan especificaciones detalladas de entrada y salida. La Figura 2.1 muestra los requisitos que se deben definir en el análisis.

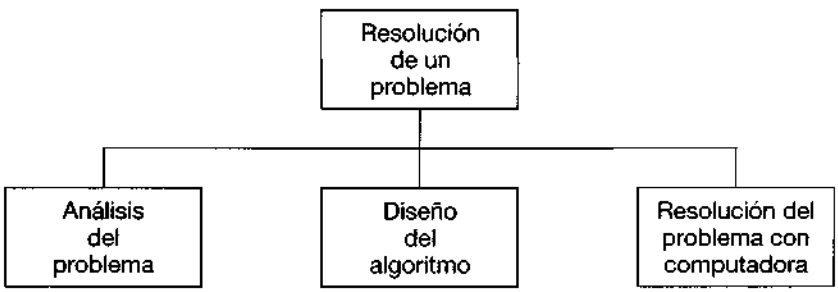


Figura 2.1. Análisis del problema.

Para poder definir bien un problema es conveniente responder a las siguientes preguntas:

- ¿Qué entradas se requieren? (tipo y cantidad).
- ¿Cuál es la salida deseada? (tipo y cantidad).
- ¿Qué método produce la salida deseada?

Problema 2.1

Se desea obtener una tabla con las depreciaciones acumuladas y dos valores reales de cada año de un automóvil comprado en 1.800.000 pesetas en el año 1996, durante los seis años siguientes suponiendo un valor de recuperación o rescate de 120.000. Realizar el análisis del problema, conociendo la fórmula de la depreciación anual constante D para cada año de vida útil.

$$D = \frac{\textit{coste} - \textit{valor de recuperación}}{\textit{vida útil}}$$

$$D = \frac{1.800.000 - 120.000}{6} = \frac{1.680.000}{6} = 280.000$$

Entrada
- coste original
- vida útil
- valor de recuperación

Salida
- depreciación anual por año
- depreciación acumulada en cada año
- valor del automóvil en cada año

Proceso
- depreciación acumulada
- cálculo de la depreciación acumulada cada año
- cálculo del valor del automóvil en cada año

La Tabla 2.1 muestra la salida solicitada.

Tabla 2.1.

Año	Depreciación	Depreción acumulada	Valor anual
1 (1996)	280.000	280.000	1.520.000
2 (1997)	280.000	560.000	1.240.000
3 (1998)	280.000	840.000	960.000
4 (1999)	280.000	1.120.000	680.000
5 (2000)	280.000	1.400.000	400.000
6 (2001)	280.000	2.180.000	120.000

2.1.2. Diseño del algoritmo

En la etapa de análisis del proceso de programación se determina qué hace el programa. En la etapa de diseño se determina cómo hace el programa la tarea solicitada. Los métodos más eficaces para el proceso de diseño se basan en el conocido por *divide y vencerás*. Es decir, la resolución de un problema complejo se realiza dividiendo el problema en subproblemas y a continuación dividir estos subproblemas en otros de nivel más bajo, hasta que pueda ser *implementada* una solución en la computadora. Este método se conoce técnicamente como **diseño descendente** *(top-down)* o **modular**. El proceso de romper el problema en cada etapa y expresar cada paso en forma más detallada se denomina *refinamiento sucesivo*.

Cada subprograma es resuelto mediante un **módulo** *(subprograma)* que tiene un solo punto de entrada y un solo punto de salida.

Cualquier programa bien diseñado consta de un *programa principal* (el módulo de nivel más alto) que llama a subprogramas (módulos de nivel más bajo) que a su vez pueden llamar a otros subprogramas. Los programas estructurados de esta forma se dice que tienen un *diseño modular* y el método de romper el programa en módulos más pequeños se llama *programación modular.* Los módulos pueden ser planeados, codificados, comprobados y depurados independientemente (incluso por diferentes programadores) y a continuación combinarlos entre sí. El proceso implica la ejecución de los siguientes pasos hasta que el programa se termina:

1. Programar un módulo.
2. Comprobar el módulo.

3. Si es necesario, depurar el módulo.
4. Combinar el módulo con los módulos anteriores.

El proceso que convierte los resultados del análisis del problema en un diseño modular con refinamientos sucesivos que permitan una posterior traducción a un lenguaje se denomina **diseño del algoritmo**.

El diseño del algoritmo es independiente del lenguaje de programación en el que se vaya a codificar posteriormente.

2.1.3. Herramientas de programación

Las dos herramientas más utilizadas comúnmente para diseñar algoritmos son: *diagramas de flujo* y *pseudocódigos*.

Diagramas de flujo

Un **diagrama de flujo** *(flowchart)* es una representación gráfica de un algoritmo. Los símbolos utilizados han sido normalizados por el Instituto Norteamericano de Normalización (ANSI), y los más frecuentemente empleados se muestran en la Figura 2.2, junto con una plantilla utilizada para el dibujo de los diagramas de flujo (Fig. 2.3). En la Figura 2.4 se representa el diagrama de flujo que resuelve el Problema 2.1.

Pseudocódigo

El **pseudocódigo** es una herramienta de programación en la que las instrucciones se escriben en palabras similares al inglés o español, que facilitan tanto la escritura como la lectura de programas. En esencia, el pseudocódigo se puede definir como *un lenguaje de especificaciones de algoritmos*.

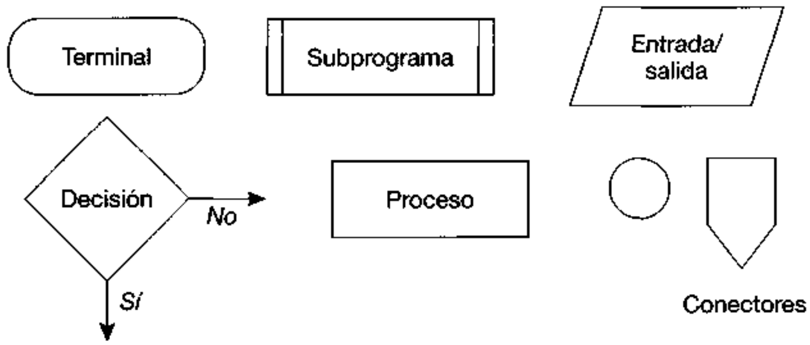


Figura 2.2. Símbolos más utilizados en los diagramas de flujo.

Aunque no existen reglas para escritura del pseudocódigo en español, se ha recogido una notación estándar que se utilizará en el libro y que ya es muy empleada en los libros de programación en español[3]. Las palabras reservadas básicas se representarán en letras negritas minúsculas. Estas palabras

[3] Para mayor ampliación sobre el *pseudocódigo*, puede consultar, entre otras, algunas de estas obras: *Fundamentos de programación*, Luis Joyanes, 2.ª edición, 1997; *Metodología de la programación*, Luis Joyanes, 1986; *Problemas de Metodología de la programación*, Luis Joyanes, 1991 (todas ellas publicadas en McGraw-Hill, Madrid), así como *Introducción a la programación*, de Clavel y Biondi, Barcelona: Masson, 1987, o bien *Introducción a la programación y a las estructuras de datos*, de Braunstein y Groia, Buenos Aires: Editorial Eudeba, 1986. Para una formación práctica puede consultar: *Fundamentos de programación: Libro de problemas*, de Luis Joyanes, Luis Rodríguez y Matilde Fernández, en McGraw-Hill (Madrid, 1998).

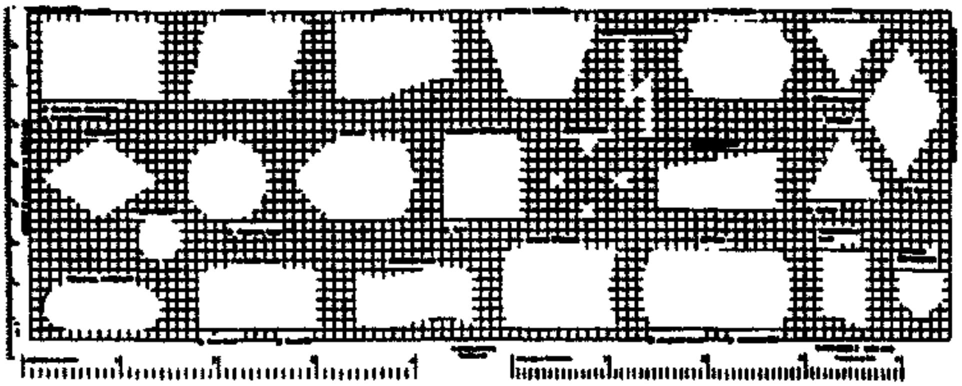


Figura 2.3. Plantilla para dibujo de diagramas de flujo.

son traducción libre de palabras reservadas de lenguajes como C, Pascal, etc. Más adelante se indicarán los pseudocódigos fundamentales a utilizar en esta obra.

El pseudocódigo que resuelve el Problema 2.1 es:

```
Previsiones de depreciacion
Introducir coste
           vida útil
           valor final de rescate (recuperación)
imprimir cabeceras
Establecer el valor inicial del Año
Calcular depreciación
mientras valor año =< vida útil hacer
   calcular depreciación acumulada
   calcular valor actual
   imprimir una línea en la tabla
   incrementar el valor del año
fin de mientras
```

y su diagrama de flujo aparece en la Figura 2.4.

Ejemplo 2.1

Calcular la paga neta de un trabajador conociendo el número de horas trabajadas, la tarifa horaria y la tasa de impuestos.

Algoritmo

```
1. Leer Horas, Tarifa, tasa
2. Calcular PagaBruta = Horas * Tarifa
3. Calcular Impuestos = PagaBruta * Tasa
4. Calcular PagaNeta = PagaBruta - Impuestos
5. Visualizar PagaBruta, Impuestos, PagaNeta
```

Ejemplo 2.2

Calcular el valor de la suma 1+2+3+...+100.

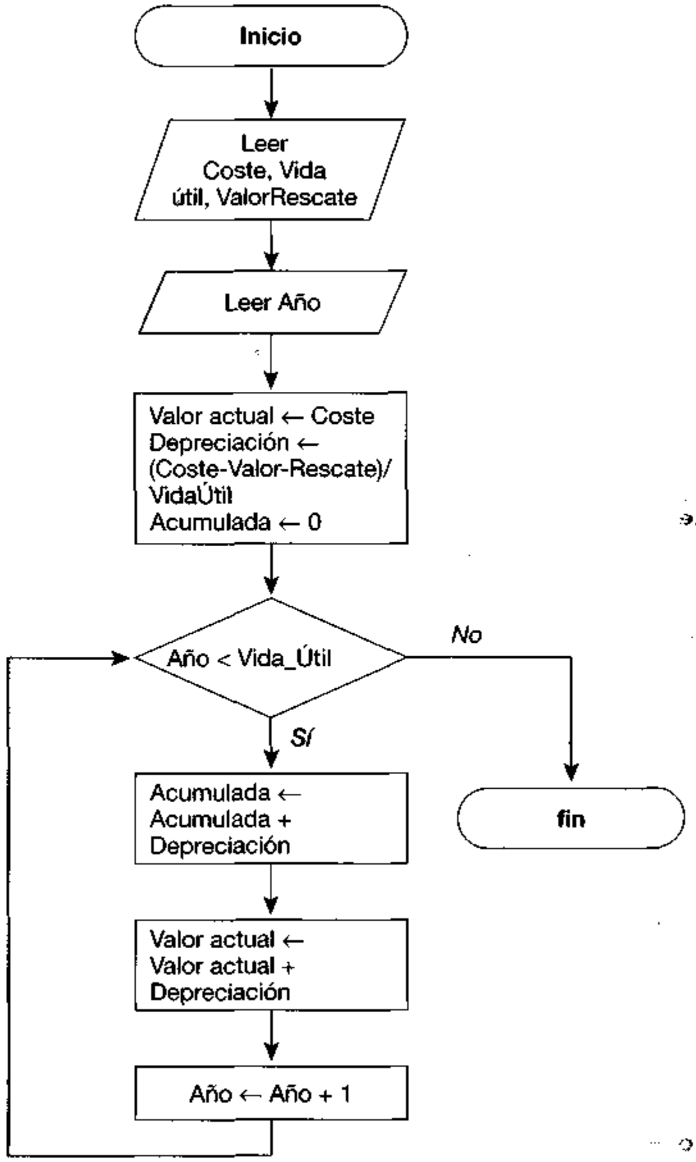


Figura 2.4. Diagrama de flujo (Problema 2.1).

Algoritmo

Se utiliza una variable `Contador` como un contador que genere los sucesivos números enteros, y Suma para almacenar las sumas parciales 1,1+2,1+2+3...

```
1. Establecer Contador a 1
2. Establecer Suma a 0
3. mientras Contador < = 100 hacer
       Sumar Contador a Suma
       Incrementar Contador en 1
   fin-mientras
4. Visualizar Suma
```

2.1.4. Codificación de un programa

Codificación es la escritura en un lenguaje de programación de la representación del algoritmo desarrollada en las etapas precedentes. Dado que el diseño de un algoritmo es independiente del lenguaje de programación utilizado para su implementación, el código puede ser escrito con igual facilidad en un lenguaje o en otro.

Para realizar la conversión del algoritmo en programa se deben sustituir las palabras reservadas en español por sus homónimos en inglés, y las operaciones/instrucciones indicadas en lenguaje natural expresarlas en el lenguaje de programación correspondiente.

```
/*
Este programa escrito en "C" obtiene una tabla de depreciaciones
acumuladas y valores reales de cada año de un determinado producto
*/
#include <stdio.h> void main()
{
   double Coste, Depreciación,
          Valor_Recuperación,
          Valor_actual,
          Acumulado,
          Valor_Anual;
   int Anio, Vida_útil;
   puts("Introduzca coste, valor recuperación y vida útil");
   scanf("%lf %lf %lf",&Coste,&Valor_Recuperación,&Vida_Útil);
   puts("Introduzca año actual");
   scanf("%d",&Anio);
   Valor_Actual = Coste;
   Depreciación = (Coste-Valor_Recuperación)/Vida_Útil;
   Acumulado = 0;
   puts("Año Depreciación Dep. Acumulada");
   while (Anio < Vida Útil)
   {
     Acumulado = Acumulado + Depreciación;
     Valor_Actual = Valor_Actual - Depreciación;
     printf("Año: %d, Depreciación:%.2lf, %.2lf Acumulada",
             Anio,Depreciación,Acumulado);
     Anio = Anio + 1;
   }
}
```

Documentación interna

Como se verá más tarde, la documentación de un programa se clasifica en *interna y externa*. La *documentación interna* es la que se incluye dentro del código del programa fuente mediante comentarios que ayudan a la comprensión del código. Todas las líneas de programas que comiencen con un símbolo / * son *comentarios*. El programa no los necesita y la computadora los ignora. Estas líneas de comentarios sólo sirven para hacer los programas más fáciles de comprender. El objetivo del programador debe ser escribir códigos sencillos y limpios.

Debido a que las máquinas actuales soportan grandes memorias (256 MB o 512 MB de memoria central mínima en computadoras personales) no es necesario recurrir a técnicas de ahorro de memoria, por lo que es recomendable que incluya el mayor número de comentarios posibles, pero, eso sí, que sean significativos.

2.1.5. Compilación y ejecución de un programa

Una vez que el algoritmo se ha convertido en un programa fuente, es preciso introducirlo en memoria mediante el teclado y almacenarlo posteriormente en un disco. Esta operación se realiza con un programa editor, posteriormente el programa fuente se convierte en un *archivo de programa* que se guarda (graba) en disco.

El **programa fuente** debe ser traducido a lenguaje máquina, este proceso se realiza con el compilador y el sistema operativo que se encarga prácticamente de la compilación.

Si tras la compilación se presentan errores (*errores de compilación*) en el programa fuente, es preciso volver a editar el programa, corregir los errores y compilar de nuevo. Este proceso se repite hasta que no se producen errores, obteniéndose el **programa objeto** que todavía no es ejecutable directamente. Suponiendo que no existen errores en el programa fuente, se debe instruir al sistema operativo para que realice la fase de **montaje** o **enlace** (*link*), carga, del programa objeto con las librerías del programa del compilador. El proceso de montaje produce un **programa ejecutable**. La Figura 2.5 describe el proceso completo de compilación/ejecución de un programa.

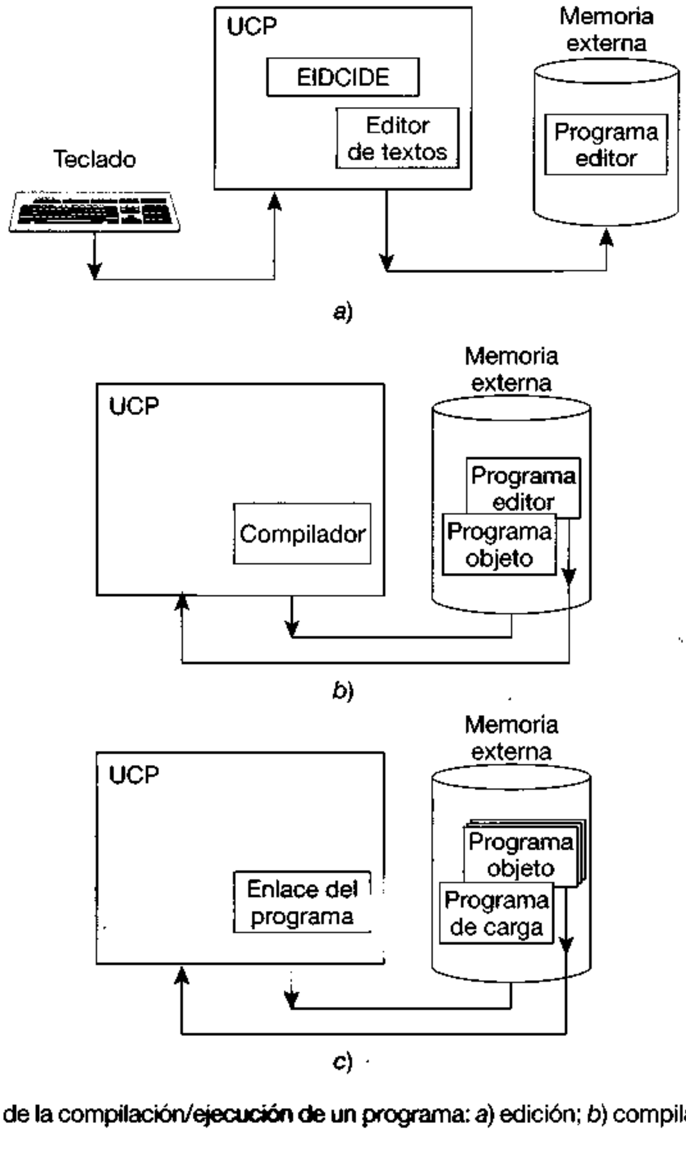


Figura 2.5. Fases de la compilación/ejecución de un programa: *a*) edición; *b*) compilación; *c*) montaje o enlace.

Cuando el programa ejecutable se ha creado, se puede ya ejecutar (correr o rodar) desde el sistema operativo con sólo teclear su nombre (en el caso de DOS). Suponiendo que no existen errores durante la ejecución (llamados **errores en tiempo de ejecución**), se obtendrá la salida de resultados del programa.

Las instrucciones u órdenes para compilar y ejecutar un programa en C puede variar según el tipo de compilador. Así el proceso de Visual C++ 6 es diferente de C bajo UNIX o bajo Linux.

2.1.6. Verificación y depuración de un programa

La *verificación* de un programa es el proceso de ejecución del programa con una amplia variedad de datos de entrada, llamados *datos de test* o *prueba*, que determinarán si el programa tiene errores («*bugs*»). Para realizar la verificación se debe desarrollar una amplia gama de datos de test: valores normales de entrada, valores extremos de entrada que comprueben los límites del programa y valores de entrada que comprueben aspectos especiales del programa.

La *depuración* es el proceso de encontrar los errores del programa y corregir o eliminar dichos errores.

Cuando se ejecuta un programa se pueden producir tres tipos de errores:

1. *Errores de compilación.* Se producen normalmente por un uso incorrecto de las reglas del lenguaje de programación y suelen ser *errores de sintaxis.* Si existe un error de sintaxis, la computadora no puede comprender la instrucción, no se obtendrá el programa objeto y el compilador imprimirá una lista de todos los errores encontrados durante la compilación.
2. *Errores de ejecución.* Estos errores se producen por instrucciones que la computadora puede comprender pero no ejecutar. Ejemplos típicos son: división por cero y raíces cuadradas de números negativos. En estos casos se detiene la ejecución del programa y se imprime un mensaje de error.
3. *Errores lógicos.* Se producen en la lógica del programa y la fuente del error suele ser el diseño del algoritmo. Estos errores son los más difíciles de detectar, ya que el programa puede funcionar y no producir errores de compilación ni de ejecución, y sólo puede advertir el error por la obtención de resultados incorrectos. En este caso se debe volver a la fase de diseño del algoritmo, modificar el algoritmo, cambiar el programa fuente y compilar y ejecutar una vez más.

2.1.7. Documentación y mantenimiento

La documentación de un problema consta de las descripciones de los pasos a dar en el proceso de resolución de un problema. La importancia de la documentación debe ser destacada por su decisiva influencia en el producto final. Programas pobremente documentados son difíciles de leer, más difíciles de depurar y casi imposibles de mantener y modificar.

La documentación de un programa puede ser *interna y externa.* La *documentación interna* es la contenida en líneas de comentarios. La *documentación externa* incluye análisis, diagramas de flujo y/o pseudocódigos, manuales de usuario con instrucciones para ejecutar el programa y para interpretar los resultados.

La documentación es vital cuando se desea corregir posibles errores futuros o bien cambiar el programa. Tales cambios se denominan *mantenimiento del programa.* Después de cada cambio la documentación debe ser actualizada para facilitar cambios posteriores. Es práctica frecuente numerar las sucesivas versiones de los programas **1.0**, **1.1**, **2.0**, **2.1**, etc. (Si los cambios introducidos son importantes, se varía el primer dígito [**1.0**, **2.0**,...], en caso de pequeños cambios sólo se varía el segundo dígito [**2.0**, **2.1**...].)